université de BORDEAUX

Inria
INVENTEURS DU MONDE NUMÉRIQUE

The StarPU Runtime System @ Exascale ?

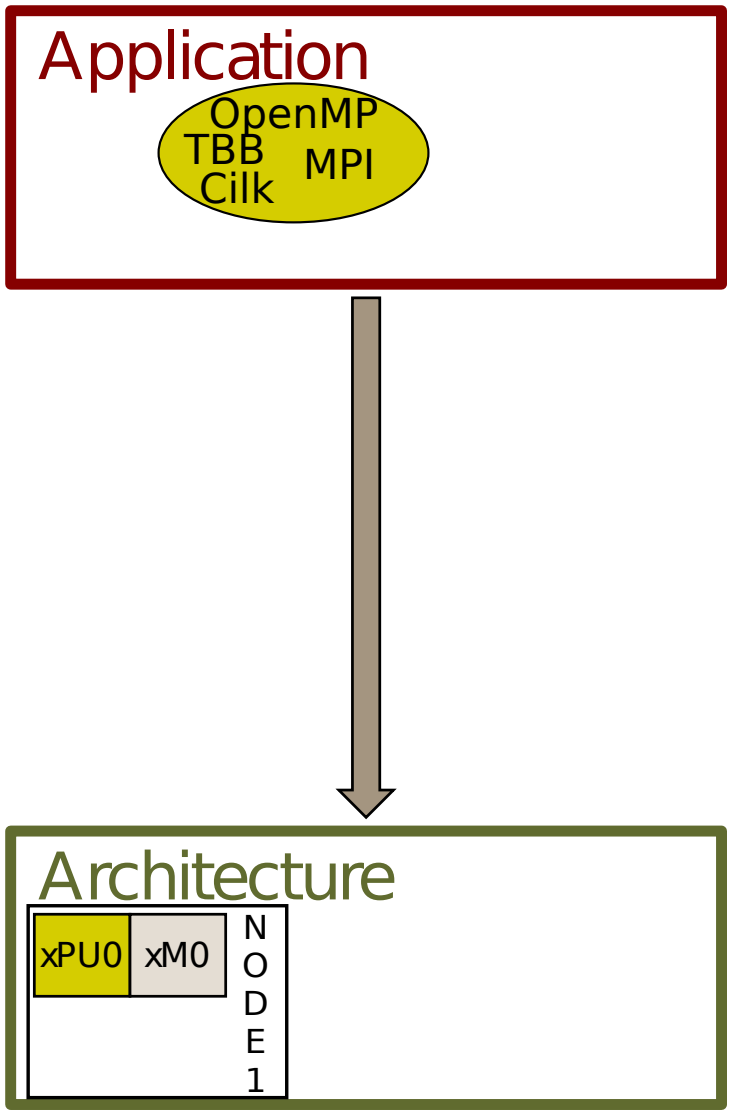
Scheduling and Programming over Upcoming Machines

RESPA SC16 Workshop, Salt Lake City, November 18, 2016

Terry Cojean, STORM TEAM
INRIA Bordeaux Sud-Ouest
Université de Bordeaux

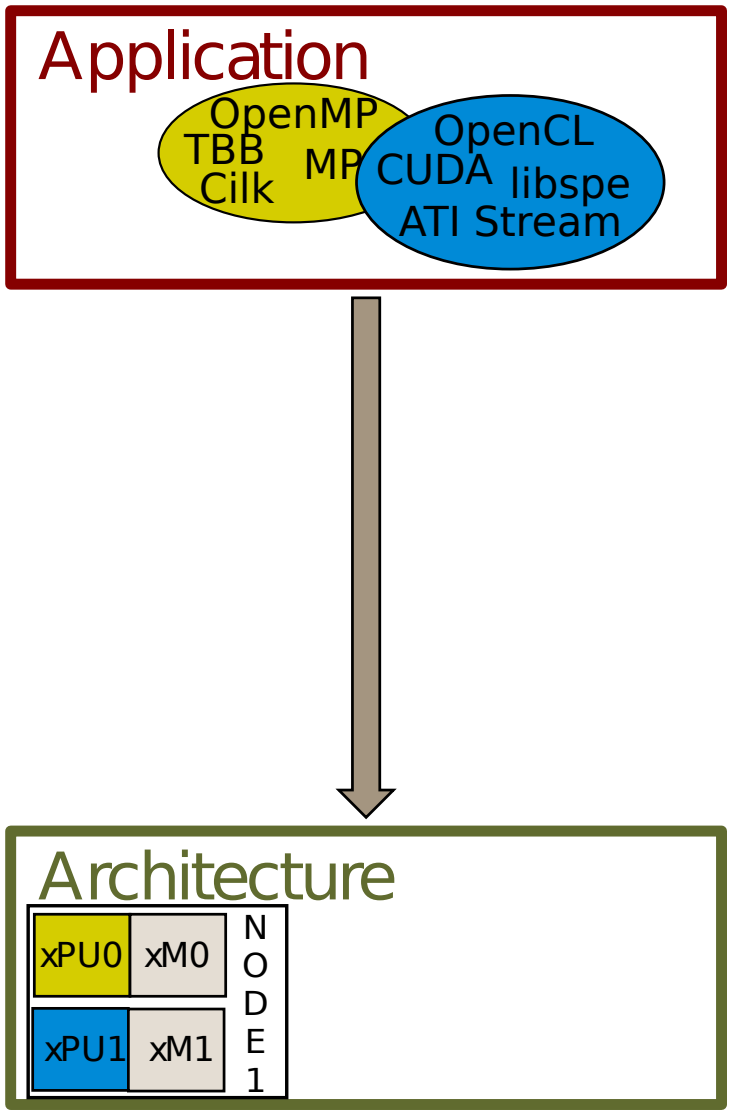
Outline

Application programming



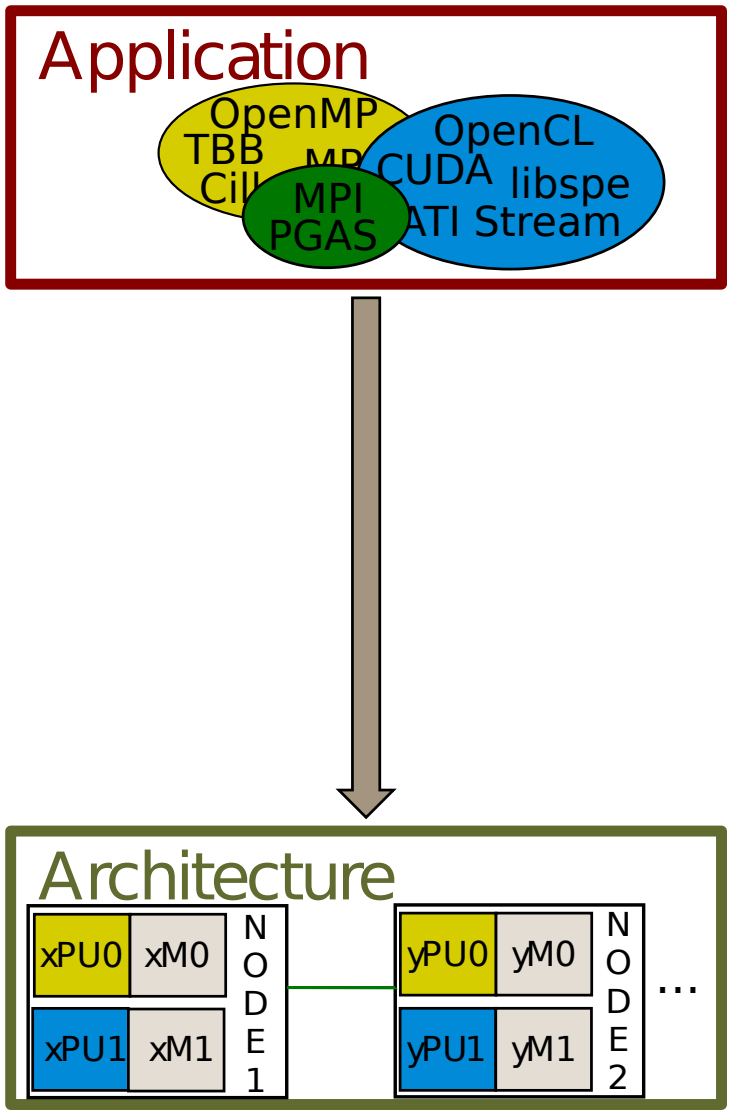
- The classical approach is based on a mixture of technologies (e.g., OpenMP)

Application programming



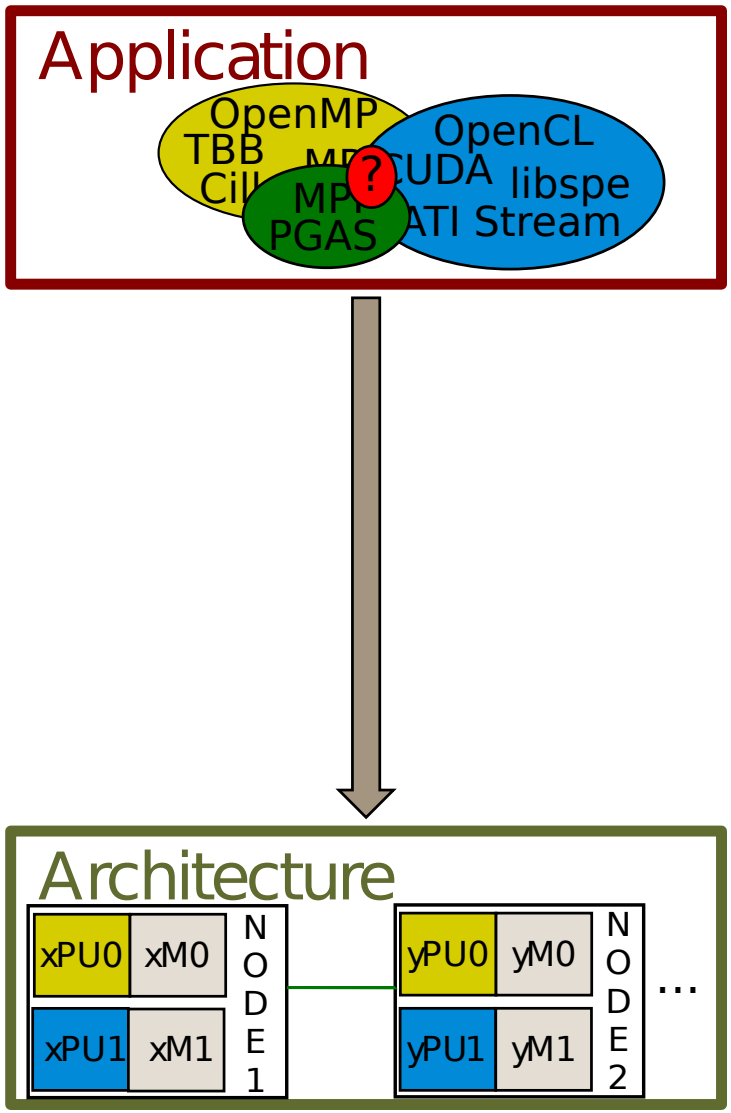
- The classical approach is based on a mixture of technologies (e.g., OpenMP+CUDA)

Application programming



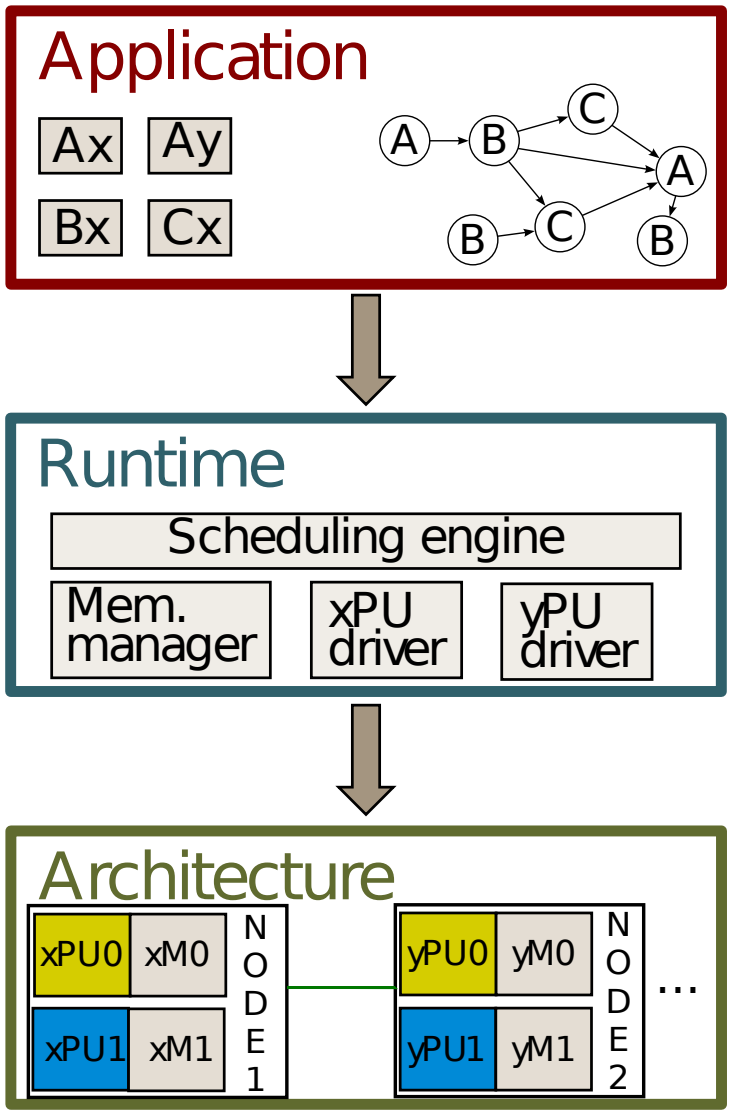
- The classical approach is based on a mixture of technologies (e.g., OpenMP+CUDA+MPI)

Application programming



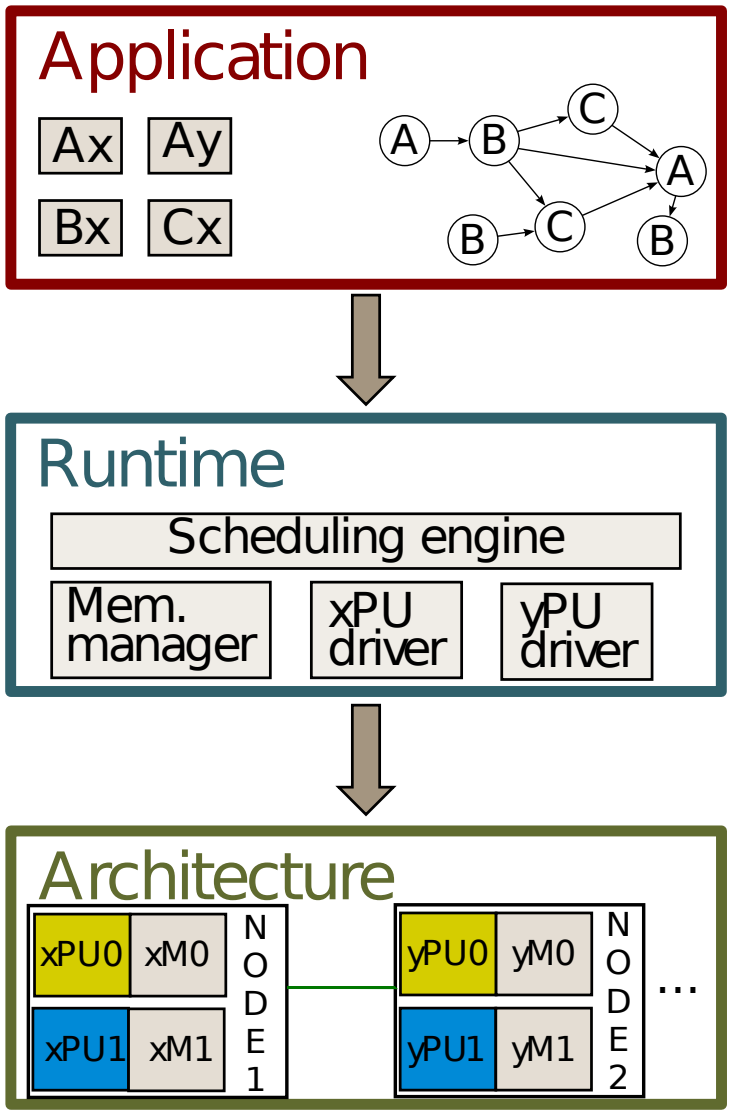
- The classical approach is based on a mixture of technologies (e.g., OpenMP+CUDA+MPI)
  - requires a big programming effort.
  - is prone to (performance) portability issues.

Application programming



- The classical approach is based on a mixture of technologies (e.g., OpenMP+CUDA+MPI)
  - requires a big programming effort.
  - is prone to (performance) portability issues.
- runtimes provide an abstraction layer that hides the architecture details.
- the workload is expressed as a DAG (Directed Acyclic Graph) of tasks scheduled by the runtime.

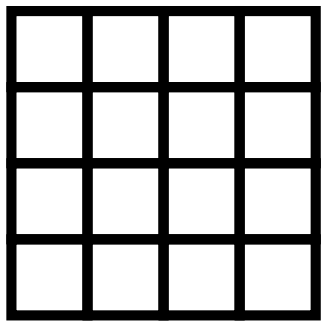
Application programming



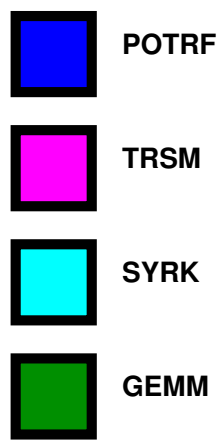
- The classical approach is based on a mixture of technologies (e.g., OpenMP+CUDA+MPI)
  - requires a big programming effort.
  - is prone to (performance) portability issues.
- runtimes provide an abstraction layer that hides the architecture details.
- the workload is expressed as a DAG (Directed Acyclic Graph) of tasks scheduled by the runtime.

StarPU: general purpose runtime system started in 2008.

Sequential Task Flow (STF) Cholesky algorithm submission

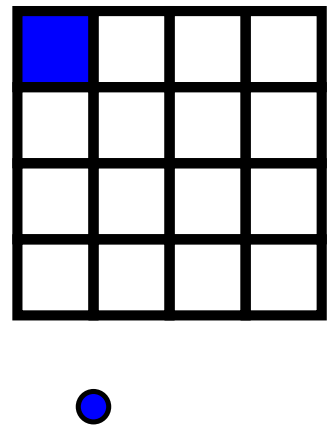


```
for (j = 0; j < N; j++) {
  submit(POTRF, A[j][j]:RW);
  for (i = j+1; i < N; i++)
    submit(TRSM, A[i][j]:RW, A[j][j]:R);
  for (i = j+1; i < N; i++) {
    submit(SYRK, A[i][i]:RW, A[i][j]:R);
    for (k = j+1; k < i; k++)
          submit(GEMM, A[i][k]:RW,
                       A[i][j]:R, A[k][j]:R);
  }
}
__wait__();
```



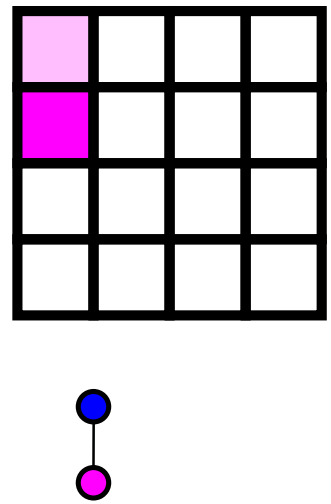
Sequential Task Flow (STF) Cholesky algorithm submission

```
for (j = 0; j < N; j++) {
  submit(POTRF, A[j][j]:RW);
  for (i = j+1; i < N; i++)
    submit(TRSM, A[i][j]:RW, A[j][j]:R);
  for (i = j+1; i < N; i++) {
    submit(SYRK, A[i][i]:RW, A[i][j]:R);
    for (k = j+1; k < i; k++)
          submit(GEMM, A[i][k]:RW,
                       A[i][j]:R, A[k][j]:R);
  }
}
__wait__();
```



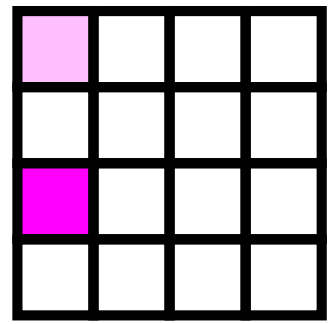
Sequential Task Flow (STF) Cholesky algorithm submission

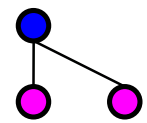
```
for (j = 0; j < N; j++) {
  submit(POTRF, A[j][j]:RW);
  for (i = j+1; i < N; i++)
    submit(TRSM, A[i][j]:RW, A[j][j]:R);
  for (i = j+1; i < N; i++) {
    submit(SYRK, A[i][i]:RW, A[i][j]:R);
    for (k = j+1; k < i; k++)
          submit(GEMM, A[i][k]:RW,
                       A[i][j]:R, A[k][j]:R);
  }
}
__wait__();
```

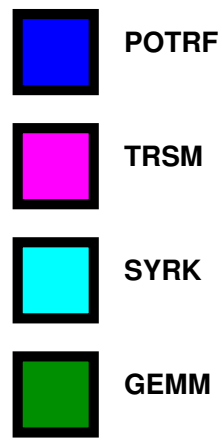


Sequential Task Flow (STF) Cholesky algorithm submission

```
for (j = 0; j < N; j++) {
  submit(POTRF, A[j][j]:RW);
  for (i = j+1; i < N; i++)
    submit(TRSM, A[i][j]:RW, A[j][j]:R);
  for (i = j+1; i < N; i++) {
    submit(SYRK, A[i][i]:RW, A[i][j]:R);
    for (k = j+1; k < i; k++)
          submit(GEMM, A[i][k]:RW,
                       A[i][j]:R, A[k][j]:R);
  }
}
__wait__();
```

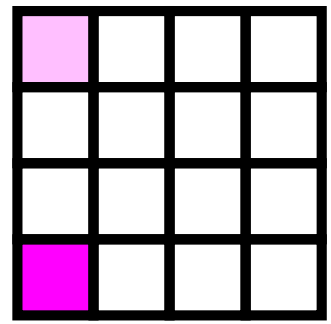


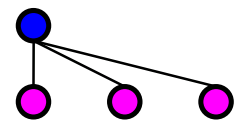


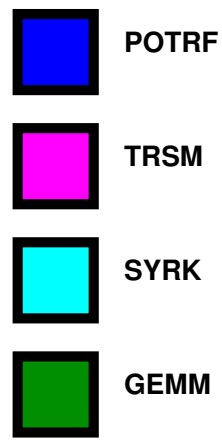


Sequential Task Flow (STF) Cholesky algorithm submission

```
for (j = 0; j < N; j++) {
  submit(POTRF, A[j][j]:RW);
  for (i = j+1; i < N; i++)
    submit(TRSM, A[i][j]:RW, A[j][j]:R);
  for (i = j+1; i < N; i++) {
    submit(SYRK, A[i][i]:RW, A[i][j]:R);
    for (k = j+1; k < i; k++)
          submit(GEMM, A[i][k]:RW,
                       A[i][j]:R, A[k][j]:R);
  }
}
__wait__();
```

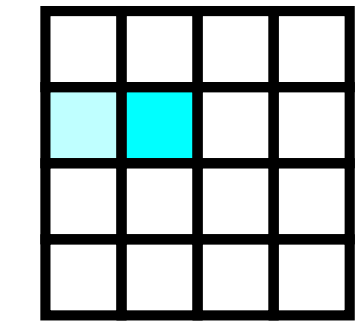


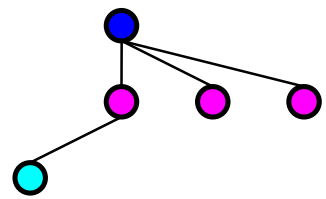


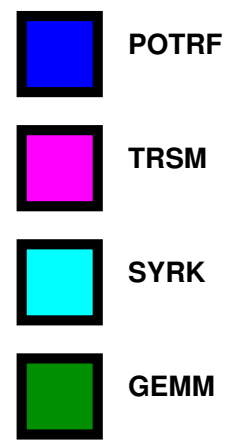


Sequential Task Flow (STF) Cholesky algorithm submission

```
for (j = 0; j < N; j++) {
  submit(POTRF, A[j][j]:RW);
  for (i = j+1; i < N; i++)
    submit(TRSM, A[i][j]:RW, A[j][j]:R);
  for (i = j+1; i < N; i++) {
    submit(SYRK, A[i][i]:RW, A[i][j]:R);
    for (k = j+1; k < i; k++)
          submit(GEMM, A[i][k]:RW,
                       A[i][j]:R, A[k][j]:R);
  }
}
__wait__();
```

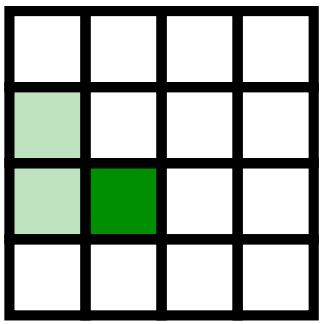


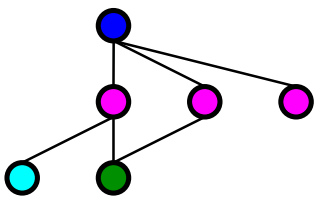


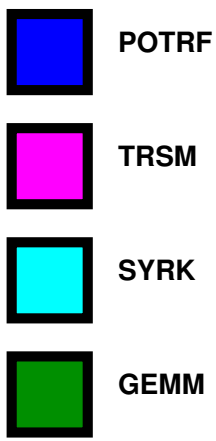


Sequential Task Flow (STF) Cholesky algorithm submission

```
for (j = 0; j < N; j++) {
  submit(POTRF, A[j][j]:RW);
  for (i = j+1; i < N; i++)
    submit(TRSM, A[i][j]:RW, A[j][j]:R);
  for (i = j+1; i < N; i++) {
    submit(SYRK, A[i][i]:RW, A[i][j]:R);
    for (k = j+1; k < i; k++)
          submit(GEMM, A[i][k]:RW,
                       A[i][j]:R, A[k][j]:R);
  }
}
__wait__();
```

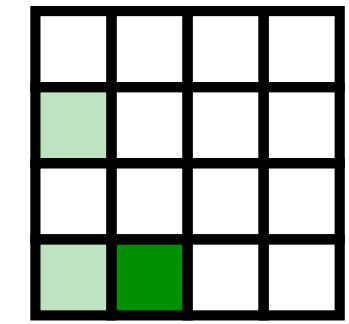


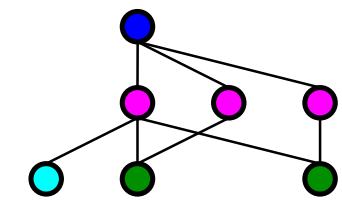


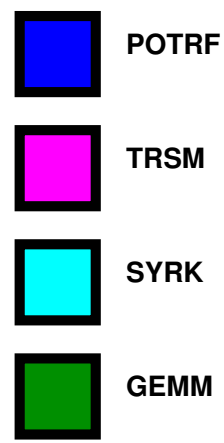


Sequential Task Flow (STF) Cholesky algorithm submission

```
for (j = 0; j < N; j++) {
  submit(POTRF, A[j][j]:RW);
  for (i = j+1; i < N; i++)
    submit(TRSM, A[i][j]:RW, A[j][j]:R);
  for (i = j+1; i < N; i++) {
    submit(SYRK, A[i][i]:RW, A[i][j]:R);
    for (k = j+1; k < i; k++)
          submit(GEMM, A[i][k]:RW,
                       A[i][j]:R, A[k][j]:R);
  }
}
__wait__();
```

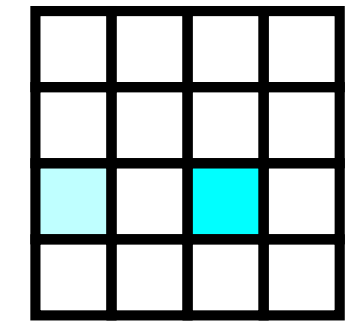


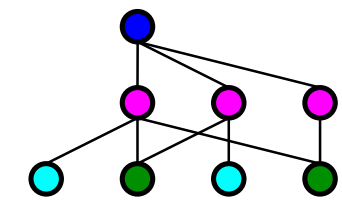


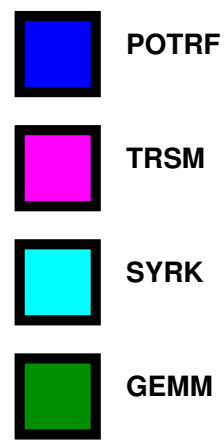


Sequential Task Flow (STF) Cholesky algorithm submission

```
for (j = 0; j < N; j++) {
  submit(POTRF, A[j][j]:RW);
  for (i = j+1; i < N; i++)
    submit(TRSM, A[i][j]:RW, A[j][j]:R);
  for (i = j+1; i < N; i++) {
    submit(SYRK, A[i][i]:RW, A[i][j]:R);
    for (k = j+1; k < i; k++)
          submit(GEMM, A[i][k]:RW,
                       A[i][j]:R, A[k][j]:R);
  }
}
__wait__();
```

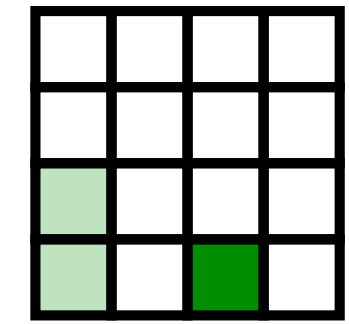


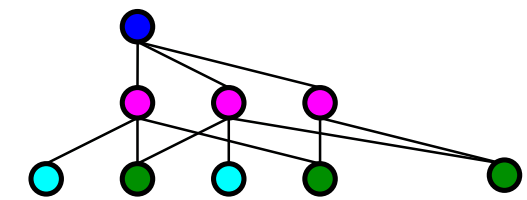


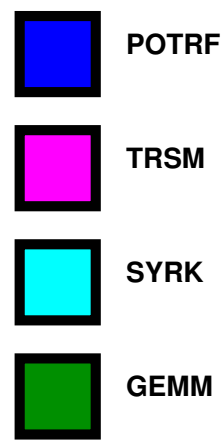


Sequential Task Flow (STF) Cholesky algorithm submission

```
for (j = 0; j < N; j++) {
  submit(POTRF, A[j][j]:RW);
  for (i = j+1; i < N; i++)
    submit(TRSM, A[i][j]:RW, A[j][j]:R);
  for (i = j+1; i < N; i++) {
    submit(SYRK, A[i][i]:RW, A[i][j]:R);
    for (k = j+1; k < i; k++)
          submit(GEMM, A[i][k]:RW,
                       A[i][j]:R, A[k][j]:R);
  }
}
__wait__();
```

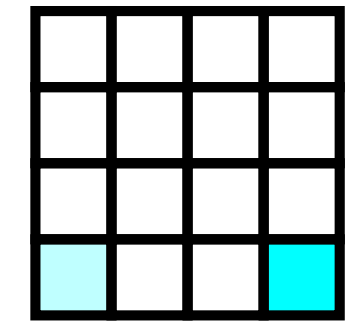


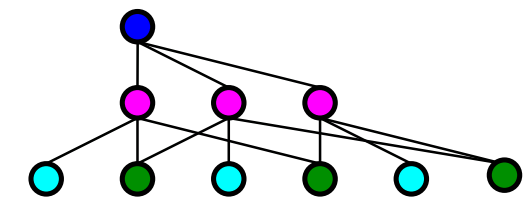


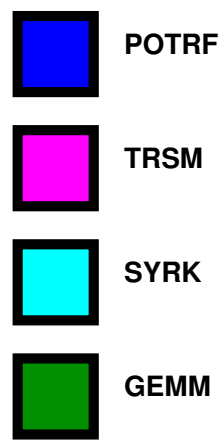


Sequential Task Flow (STF) Cholesky algorithm submission

```
for (j = 0; j < N; j++) {
  submit(POTRF, A[j][j]:RW);
  for (i = j+1; i < N; i++)
    submit(TRSM, A[i][j]:RW, A[j][j]:R);
  for (i = j+1; i < N; i++) {
    submit(SYRK, A[i][i]:RW, A[i][j]:R);
    for (k = j+1; k < i; k++)
          submit(GEMM, A[i][k]:RW,
                       A[i][j]:R, A[k][j]:R);
  }
}
__wait__();
```

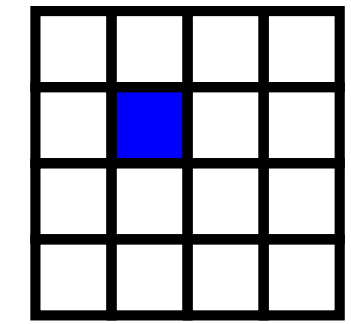


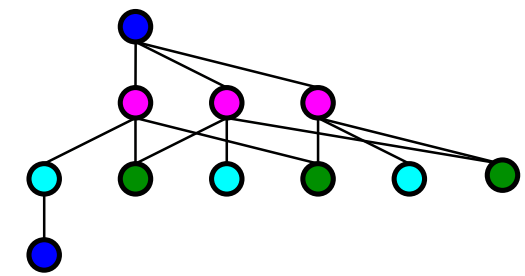


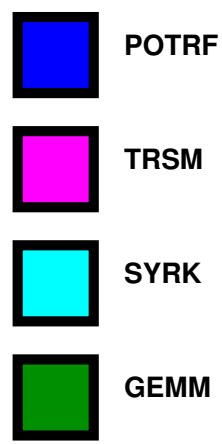


Sequential Task Flow (STF) Cholesky algorithm submission

```
for (j = 0; j < N; j++) {
  submit(POTRF, A[j][j]:RW);
  for (i = j+1; i < N; i++)
    submit(TRSM, A[i][j]:RW, A[j][j]:R);
  for (i = j+1; i < N; i++) {
    submit(SYRK, A[i][i]:RW, A[i][j]:R);
    for (k = j+1; k < i; k++)
          submit(GEMM, A[i][k]:RW,
                       A[i][j]:R, A[k][j]:R);
  }
}
__wait__();
```

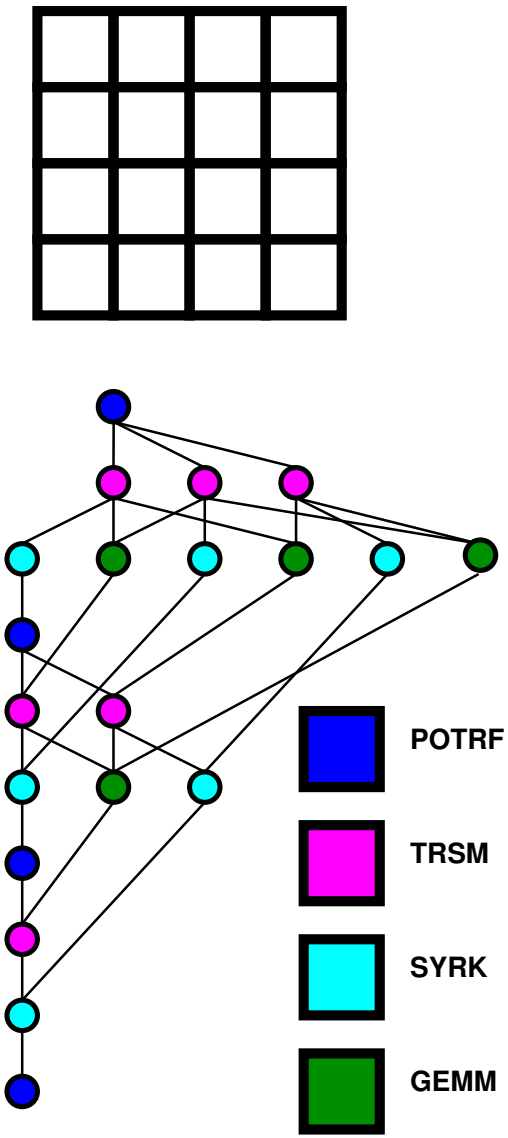




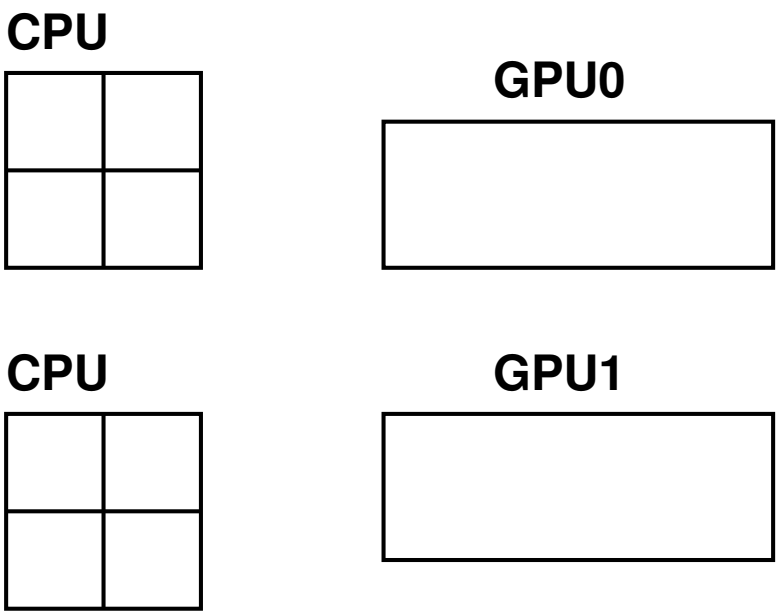


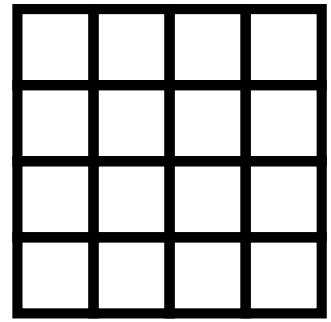
Sequential Task Flow (STF) Cholesky algorithm submission

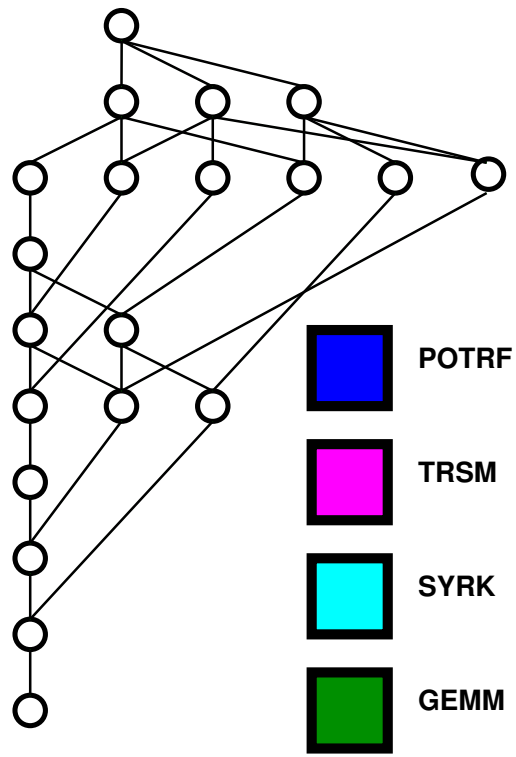
```
for (j = 0; j < N; j++) {
  submit(POTRF, A[j][j]:RW);
  for (i = j+1; i < N; i++)
    submit(TRSM, A[i][j]:RW, A[j][j]:R);
  for (i = j+1; i < N; i++) {
    submit(SYRK, A[i][i]:RW, A[i][j]:R);
    for (k = j+1; k < i; k++)
          submit(GEMM, A[i][k]:RW,
                       A[i][j]:R, A[k][j]:R);
  }
}
__wait__();
```



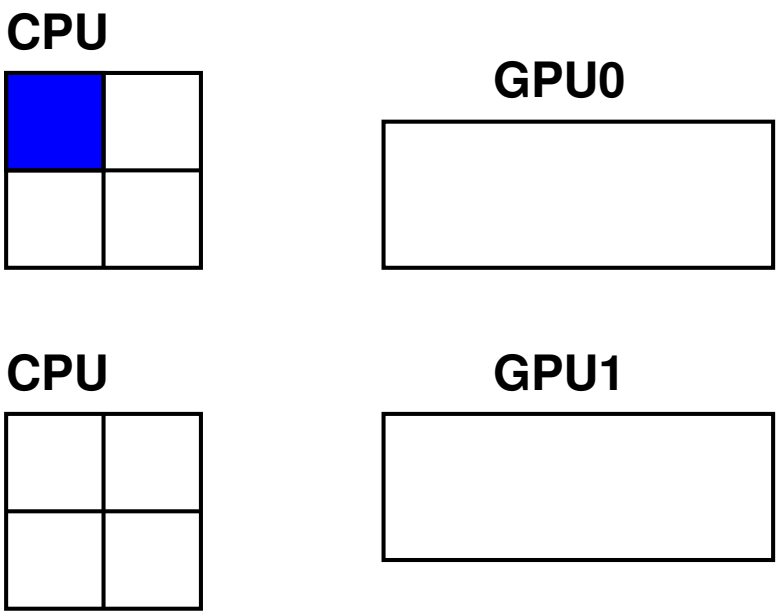
Tasks execution on a heterogeneous node

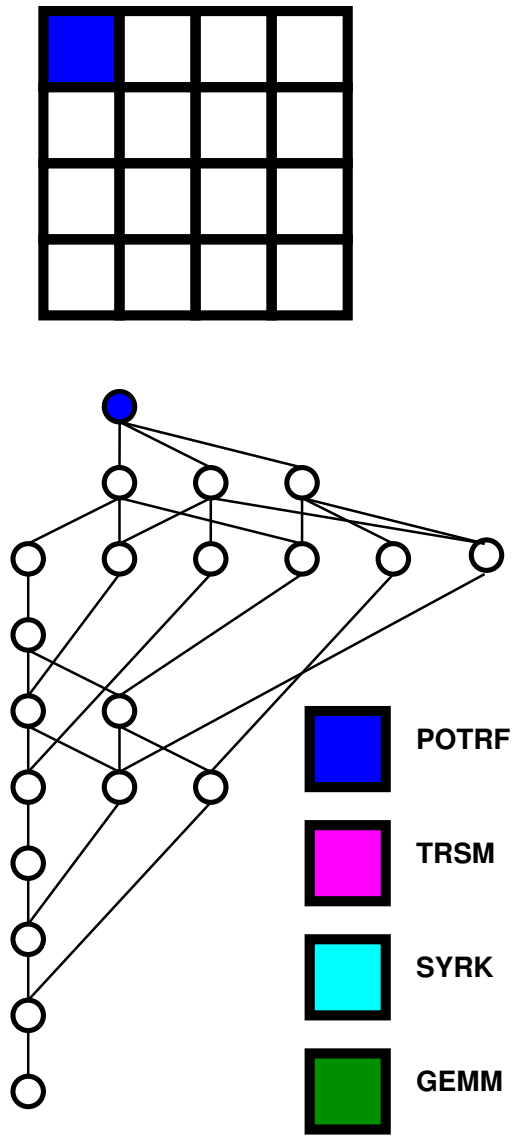




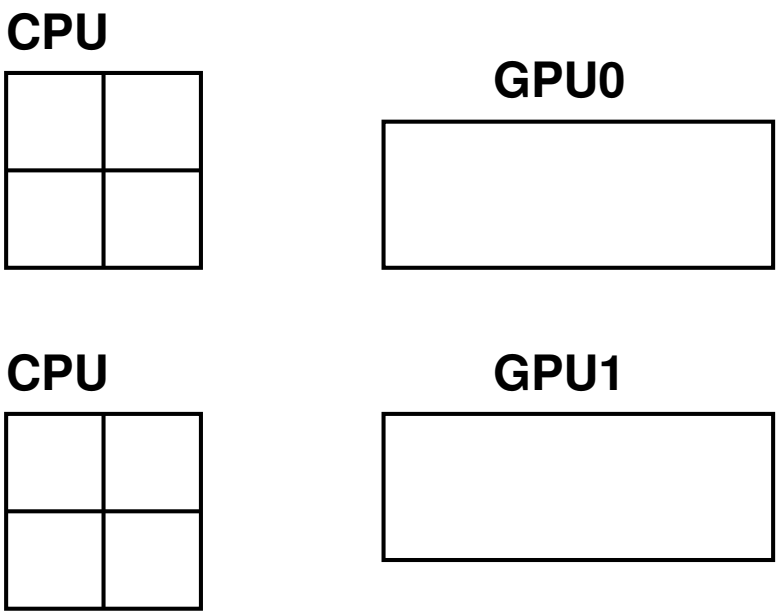


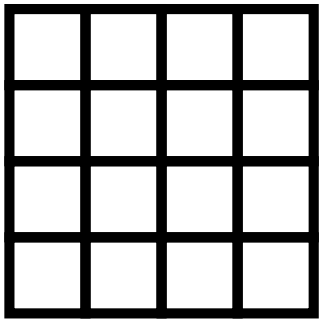
Tasks execution on a heterogeneous node

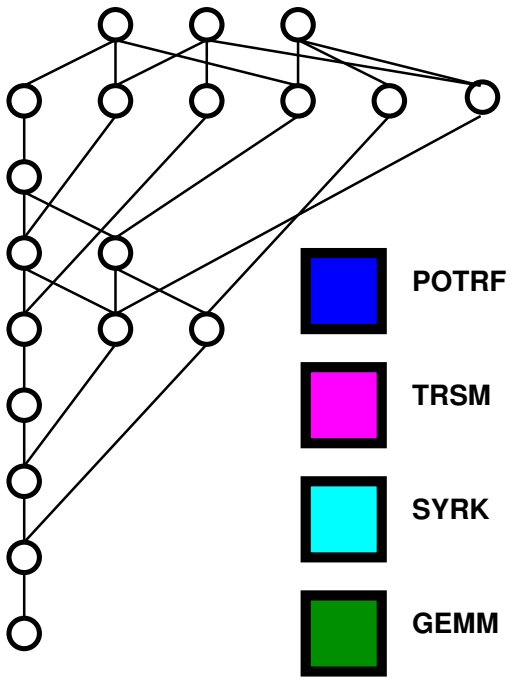




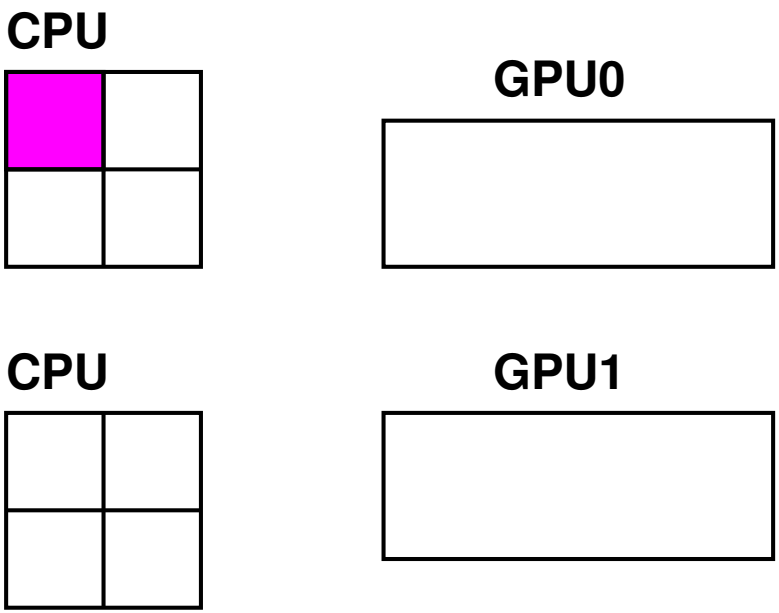
Tasks execution on a heterogeneous node

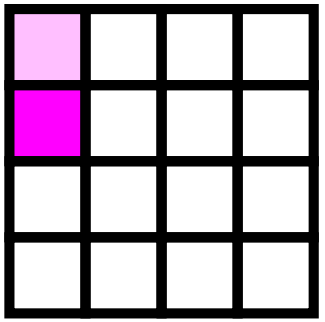


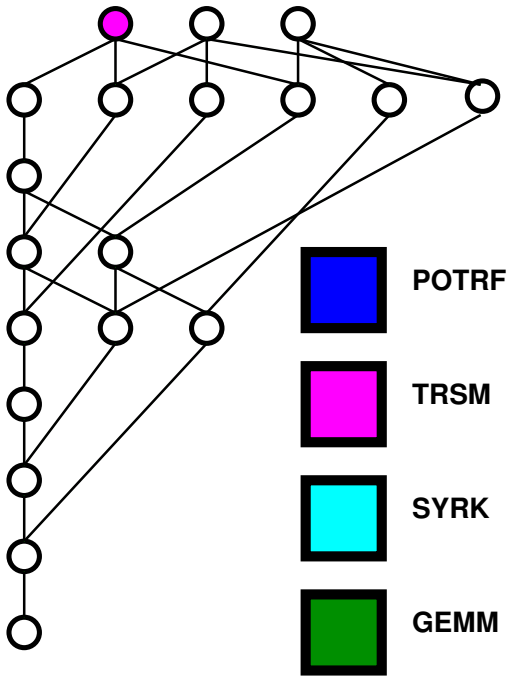




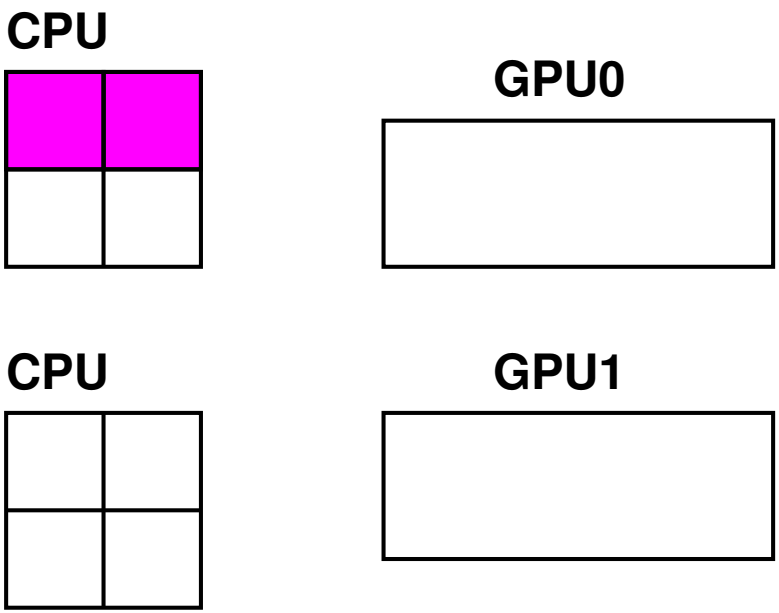
Tasks execution on a heterogeneous node

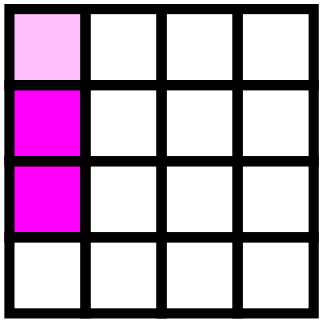


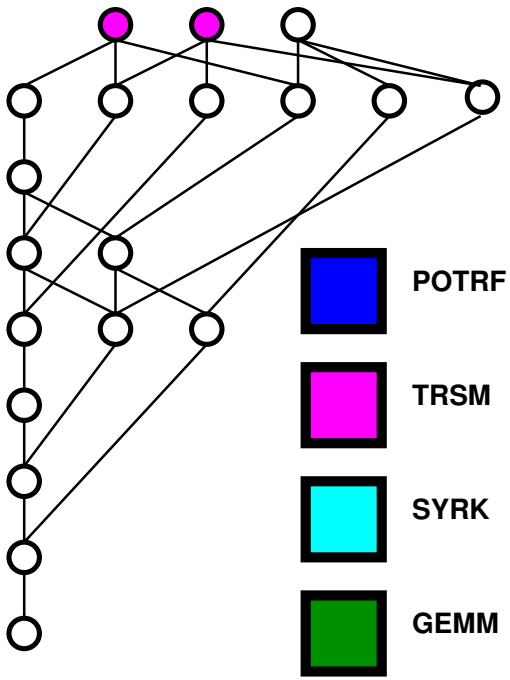




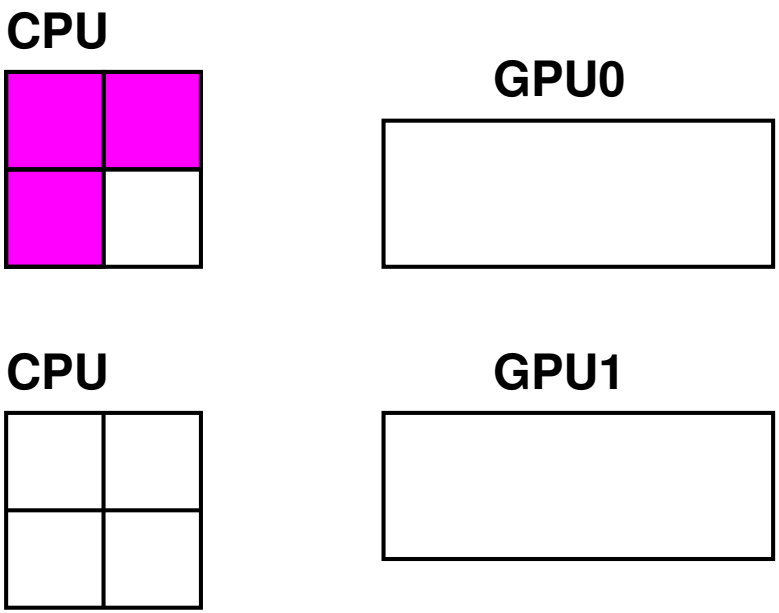
Tasks execution on a heterogeneous node

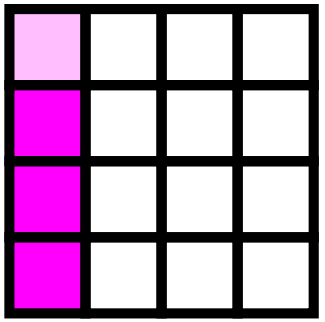


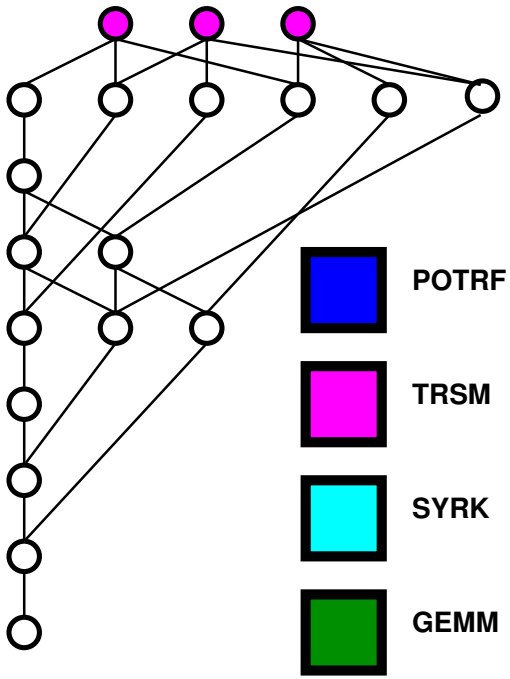




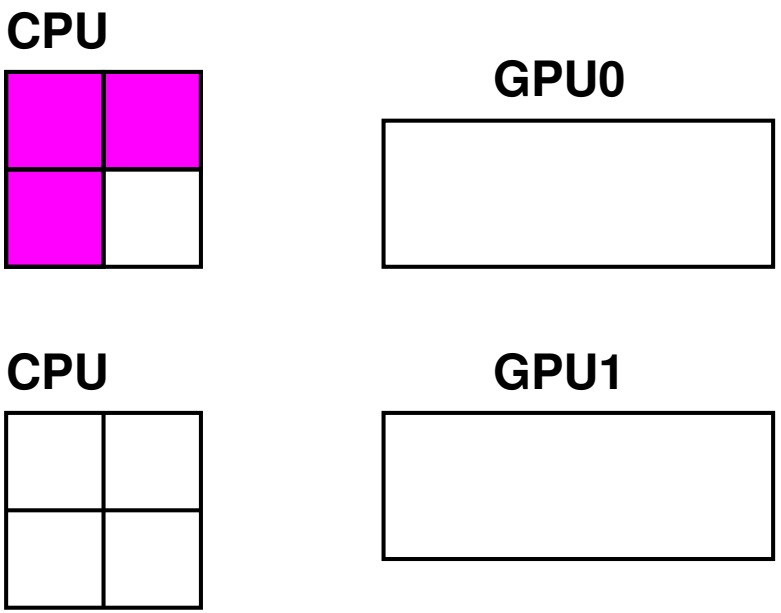
Tasks execution on a heterogeneous node

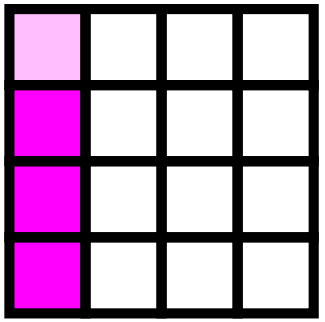




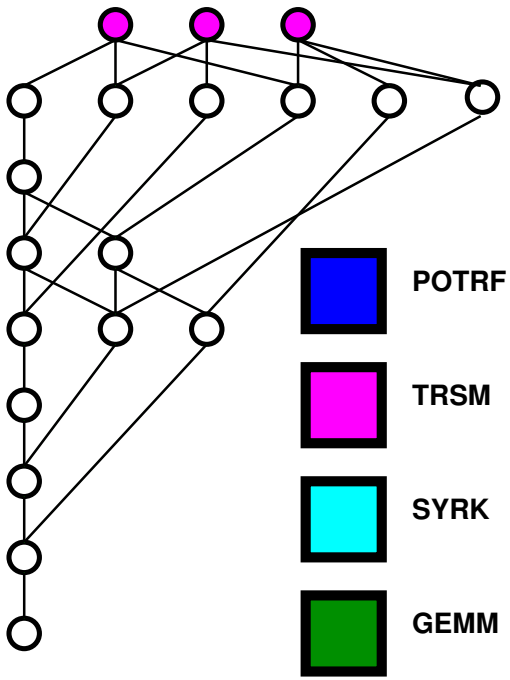


Tasks execution on a heterogeneous node

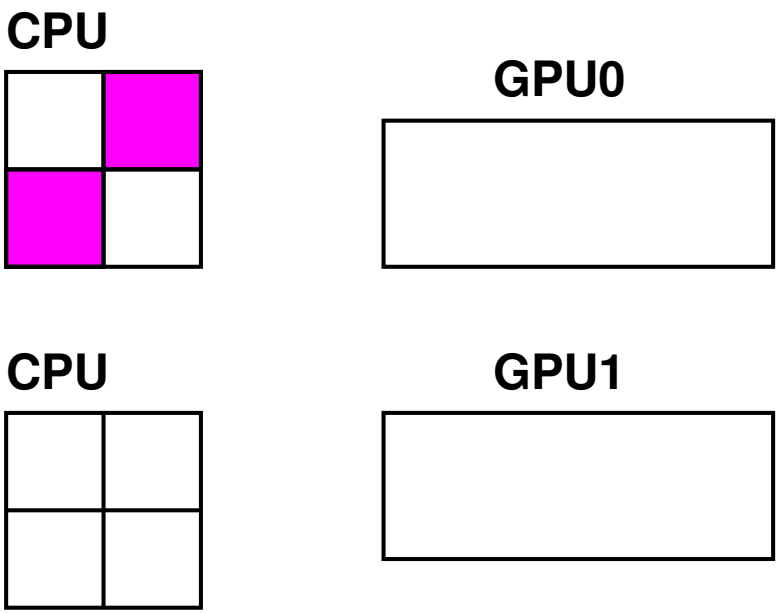


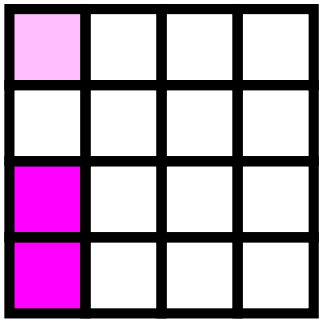


- Handles dependencies

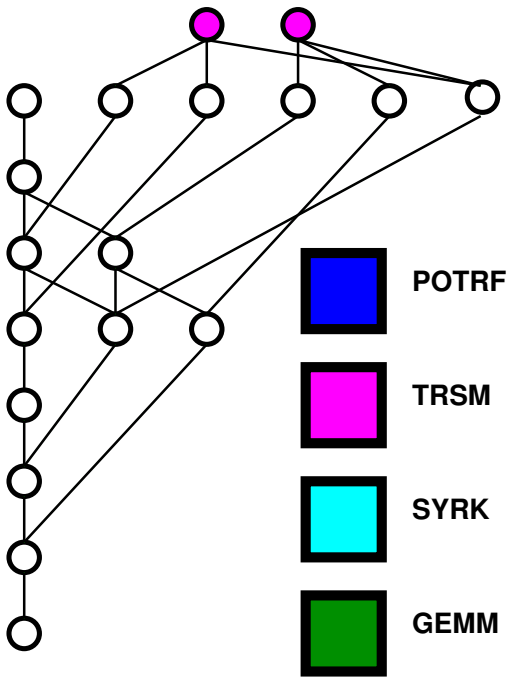


Tasks execution on a heterogeneous node

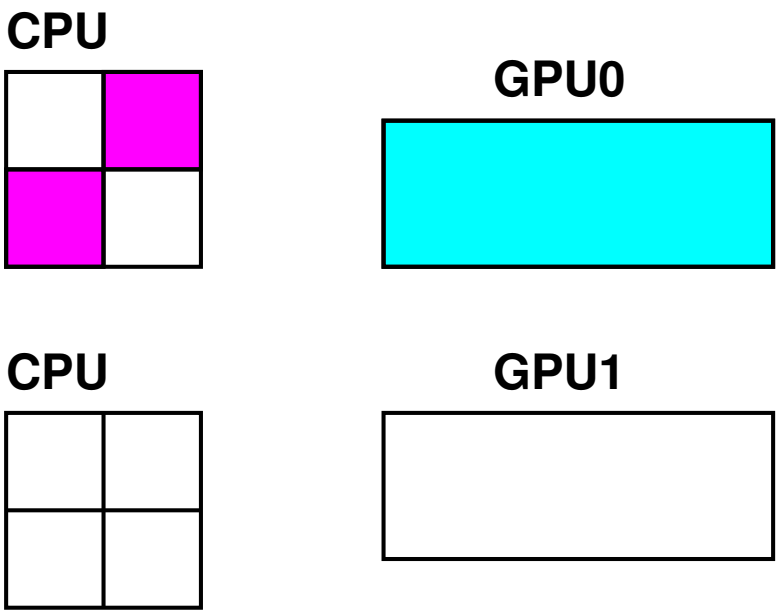


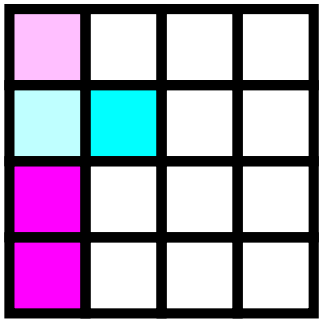


- Handles dependencies

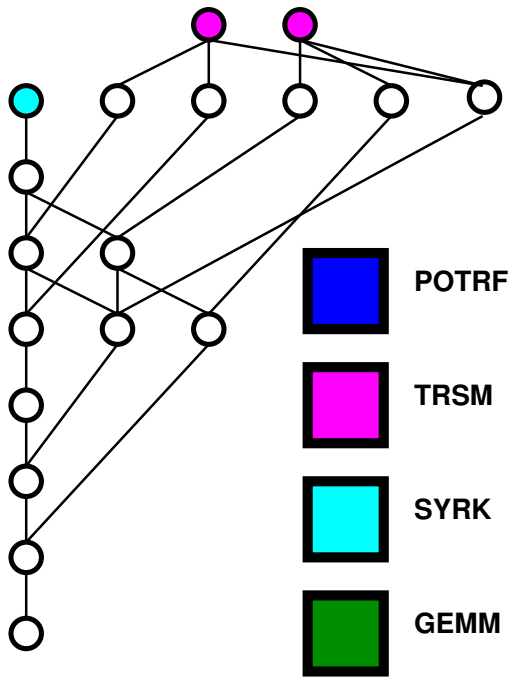


Tasks execution on a heterogeneous node

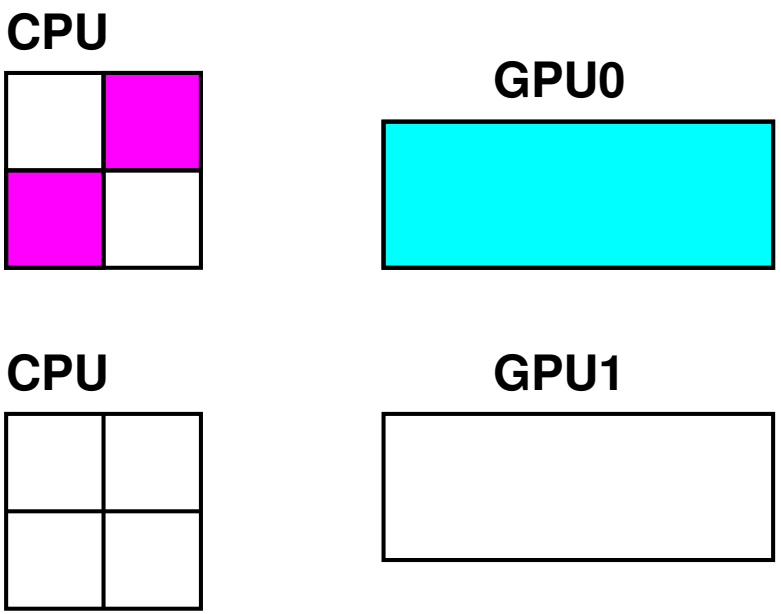


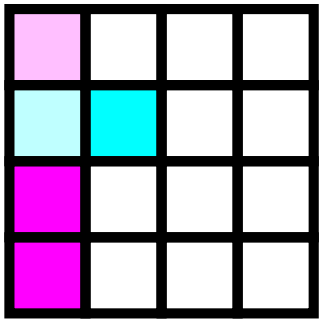


- Handles dependencies

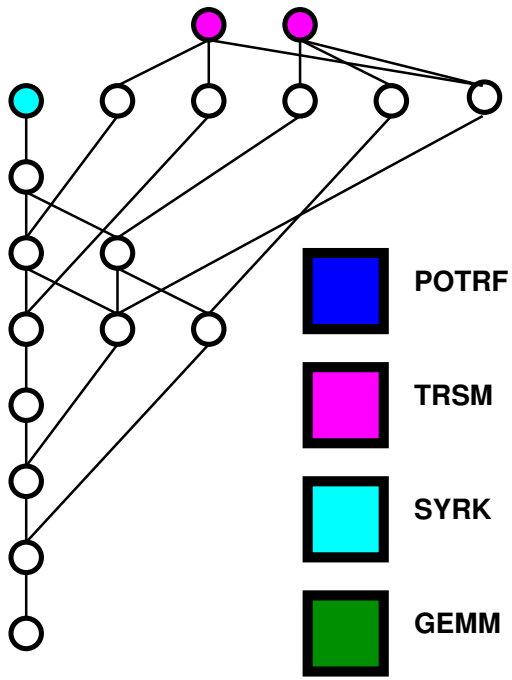


Tasks execution on a heterogeneous node

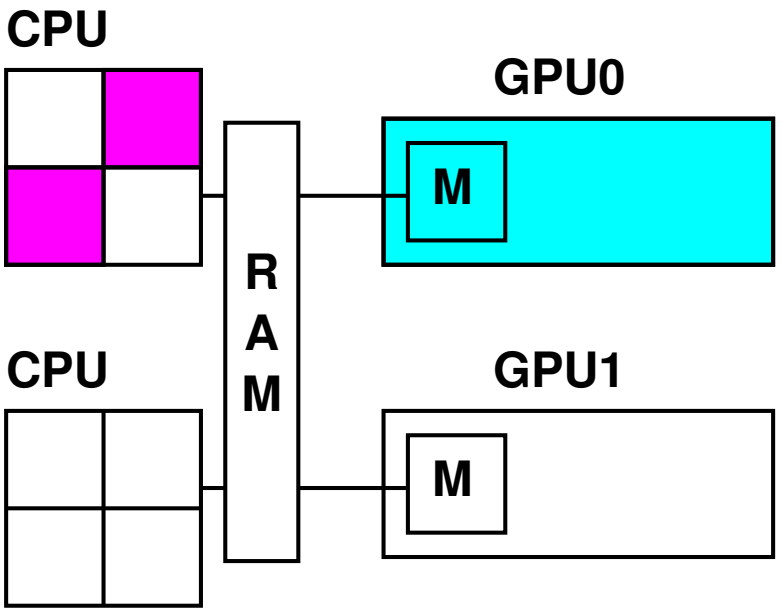


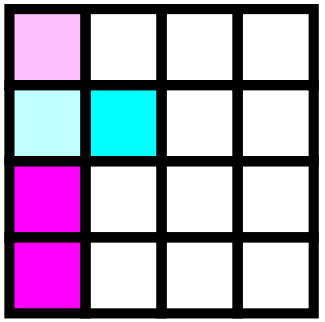


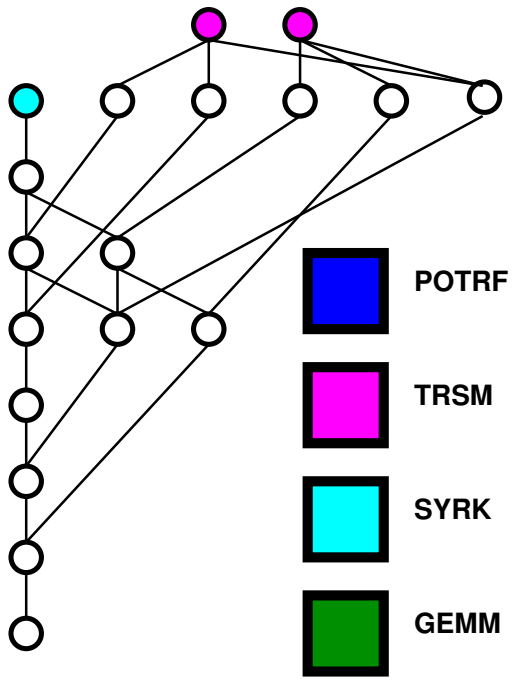
- Handles dependencies
- Handles scheduling: depends on user chosen scheduler



Tasks execution on a heterogeneous node

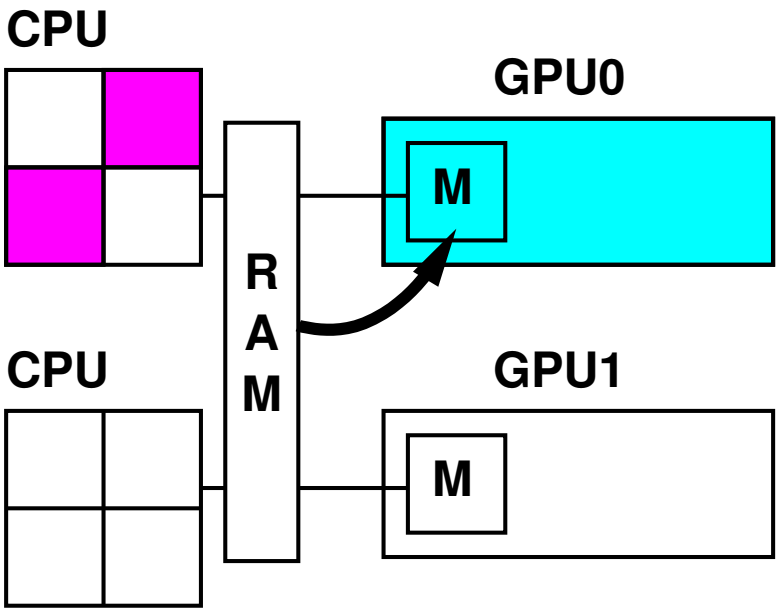


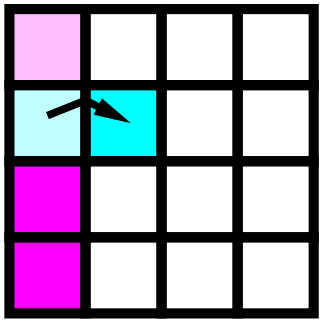


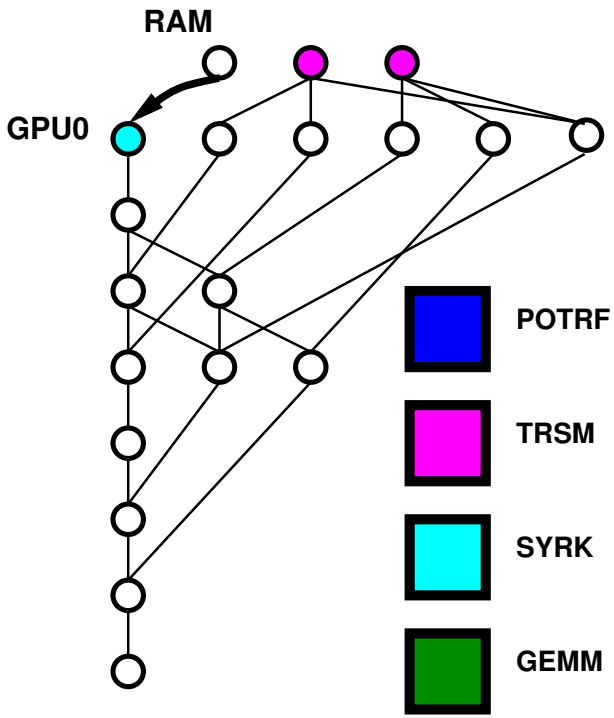


- Handles dependencies
- Handles scheduling: depends on user chosen scheduler

Tasks execution on a heterogeneous node

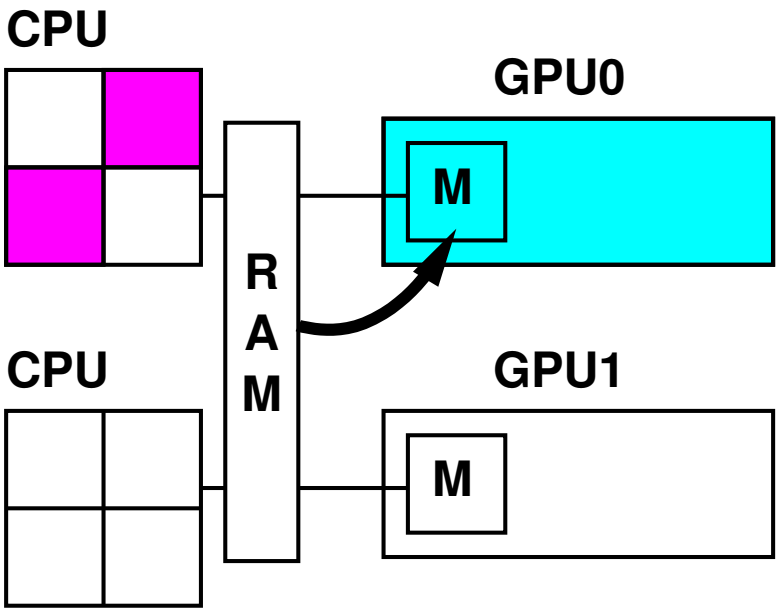


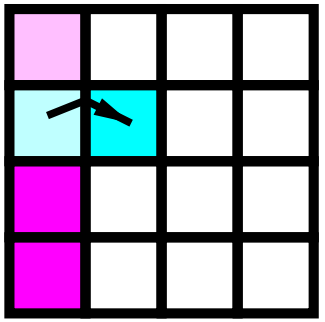


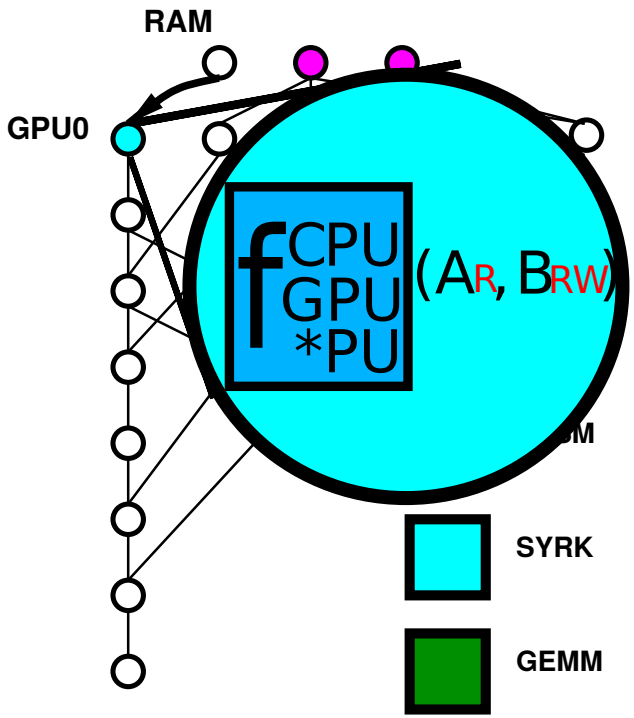


- Handles dependencies
- Handles scheduling: depends on user chosen scheduler
- Handles data consistency

Tasks execution on a heterogeneous node

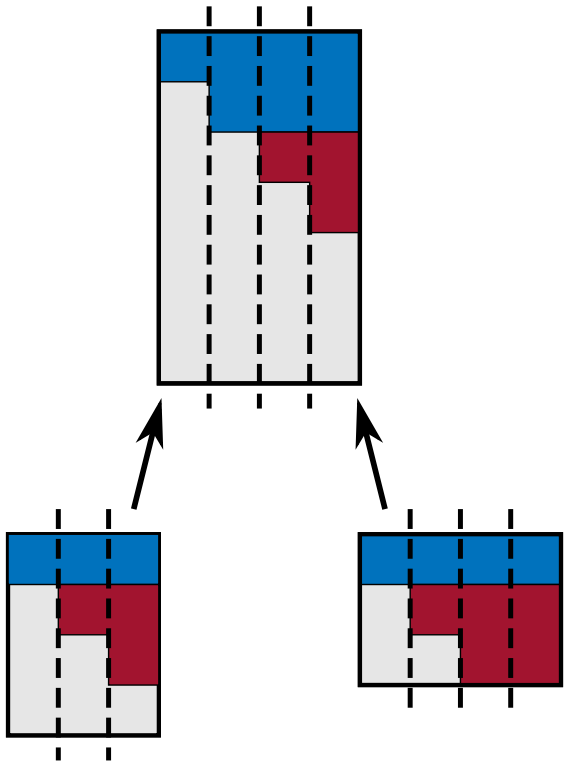






- Handles dependencies
- Handles scheduling: depends on user chosen scheduler
- Handles data consistency
- But what is a task precisely ?

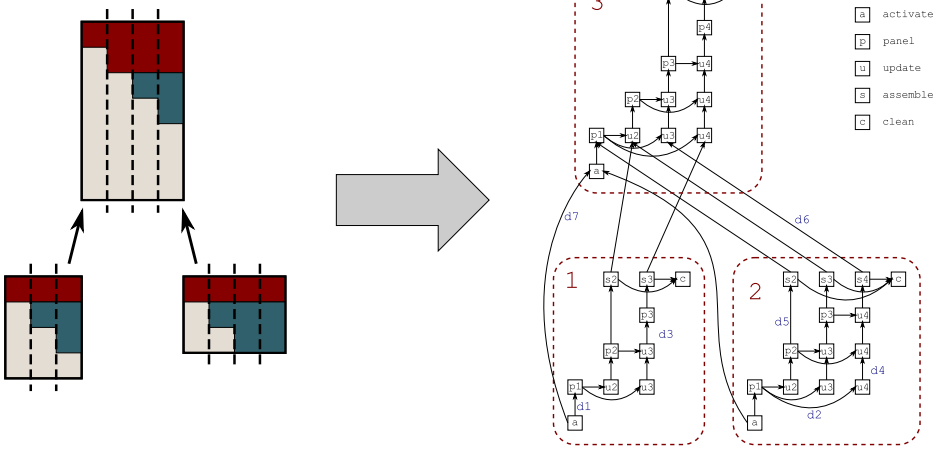
The task-based multifrontal QR factorization



```
forall fronts f in topological order
   ! allocate and initialize front
   call activate(f)

   ! front assembly
   forall children c of f
      call assemble(c, f)
      ! Deactivate child
      call deactivate(c)
   end do

   ! front factorization
   call factorize(f)
end do
```

Sequential multifrontal *QR* code with 1D block partitioning

The task-based multifrontal QR factorization



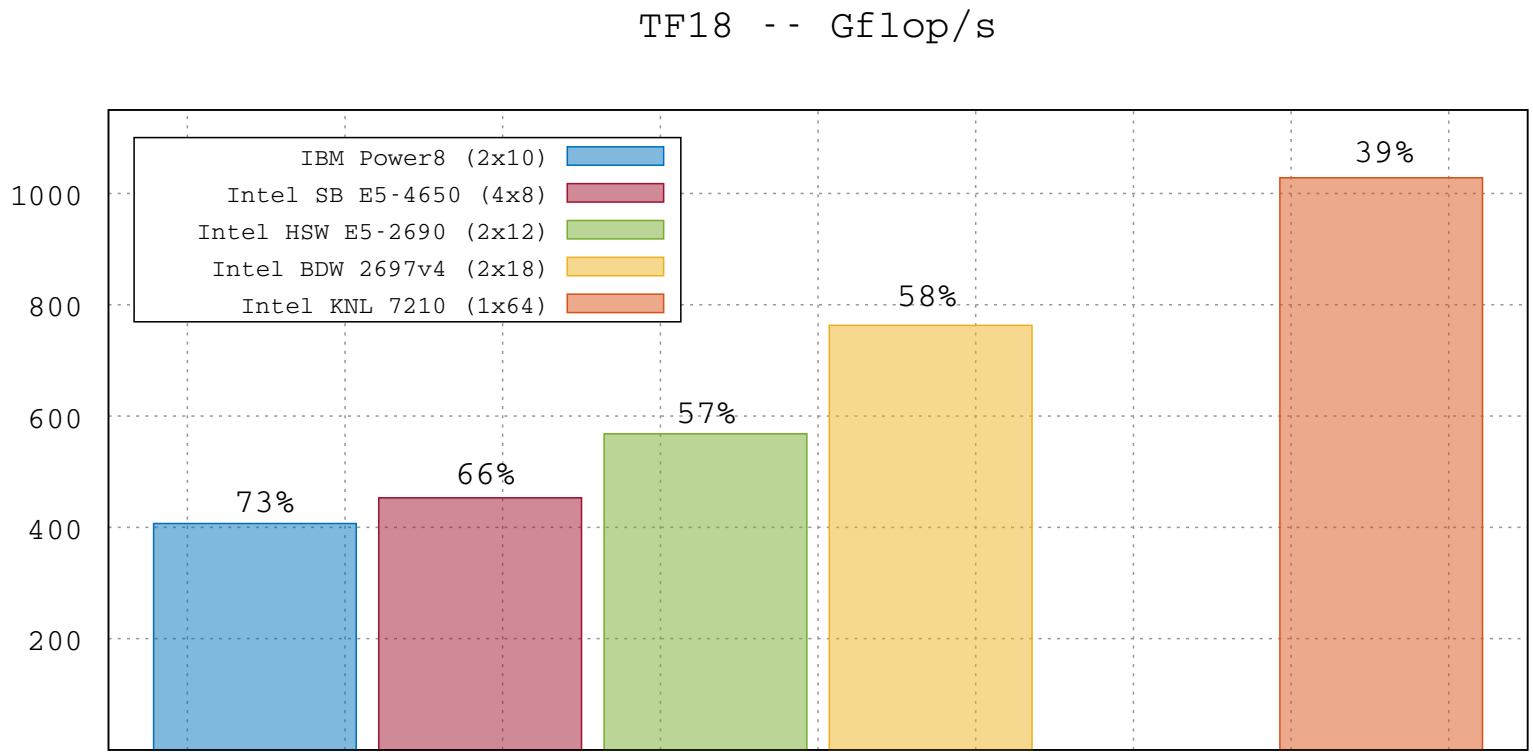

```
forall fronts f in topological order
   ! allocate and initialize front
   call submit(activate, f:RW,
      children(f):R)

   ! front assembly
   forall children c of f
      call submit(assemble, c:R, f:RW|
      C)

      call submit(deactivate, c:RW)
   end do

   ! front factorization
   call submit(factorize, f:RW)
end do

call wait_tasks_completion()
```

- STF multifrontal *QR* code with 1D block partitioning
- Elimination tree is transformed into a DAG
- Other features: 2D block partitioning, memory control, ...

Performance

PhD. F. Lopez (N7-IRIT) - led by A. Buttari (CNRS/IRIT)

StarPU schedulers used: lws (locality work stealing) and tuned HeteroPrio

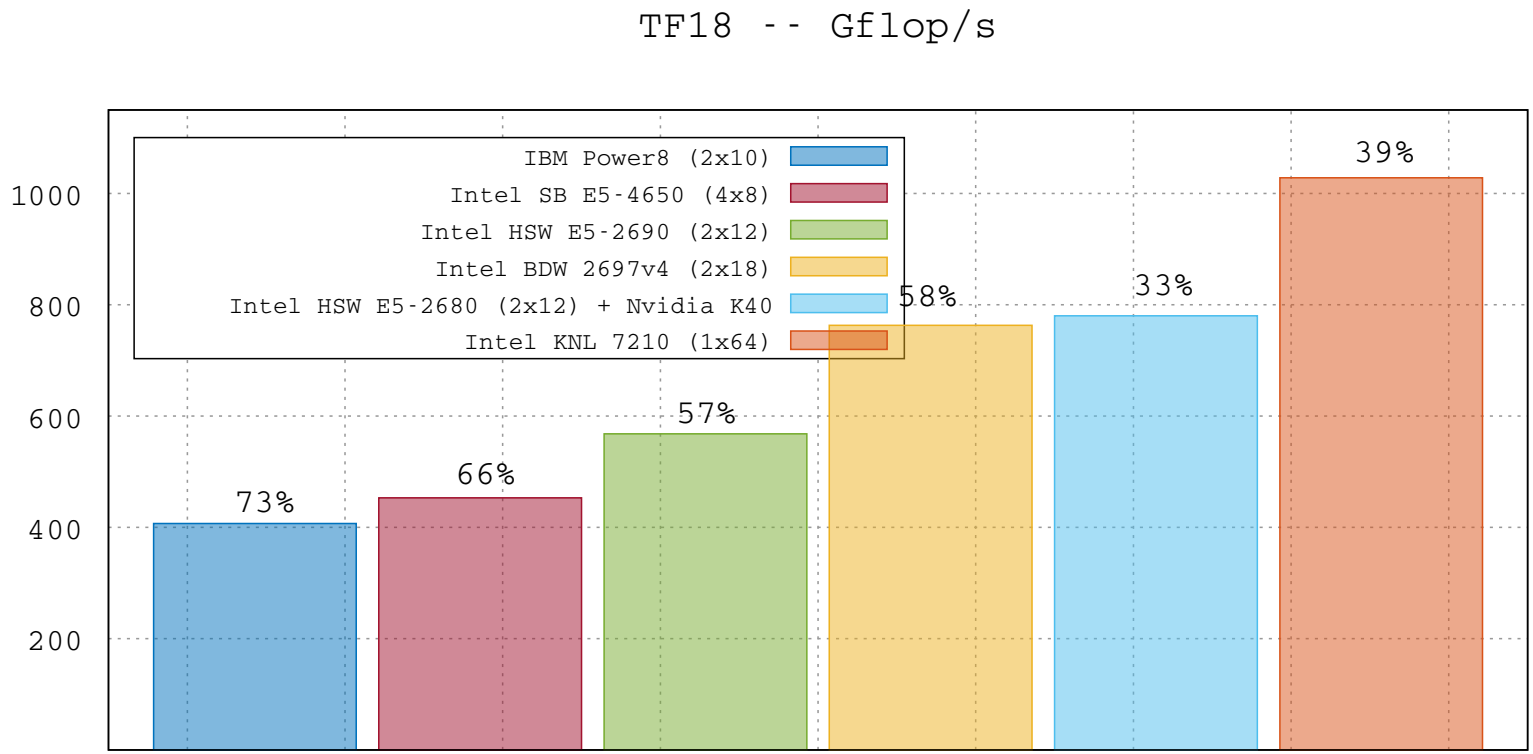


Credits: IBM, Intel, GENCI, CINES, IDRIS

Performance

PhD. F. Lopez (N7-IRIT) - led by A. Buttari (CNRS/IRIT)

StarPU schedulers used: lws (locality work stealing) and tuned HeteroPrio



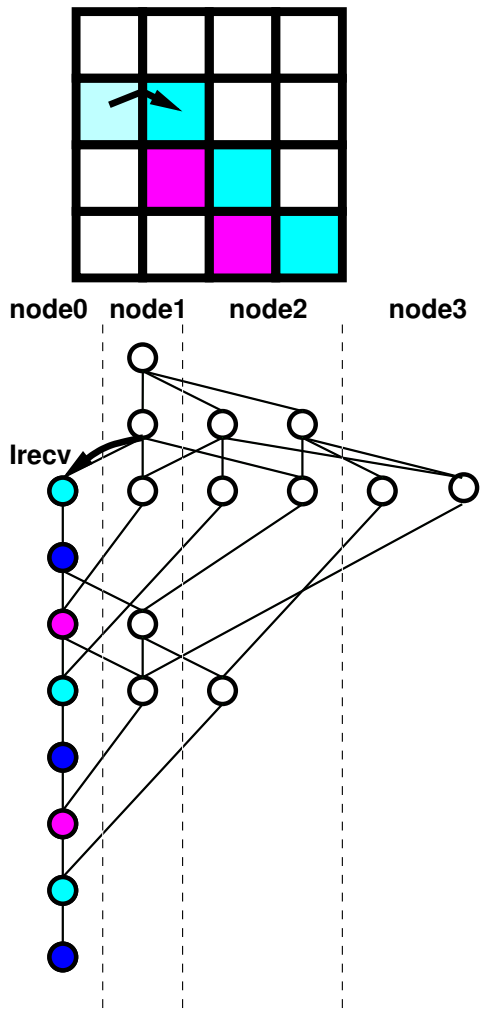
Credits: IBM, Intel, GENCI, CINES, IDRIS

StarPU-MPI example with Chameleon dense linear solver

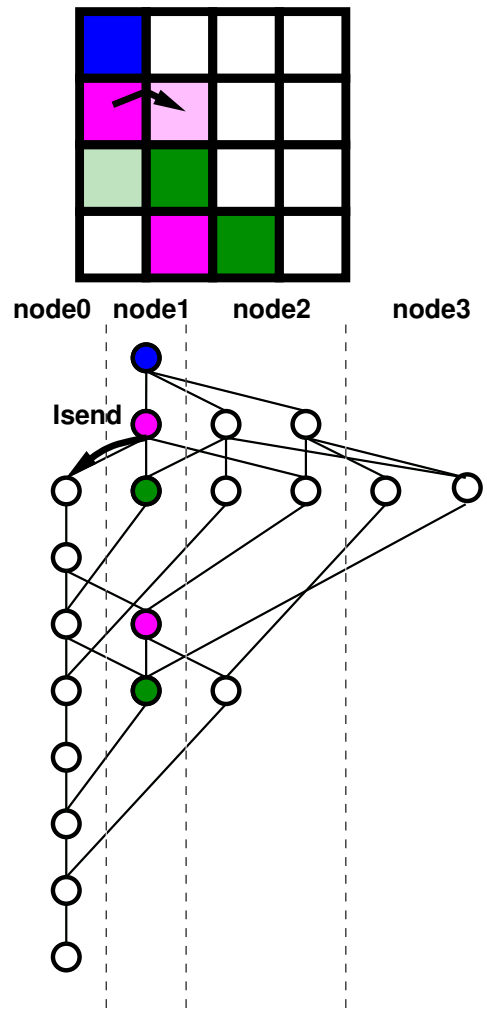
PhD Marc Sergent (Inria Storm / CEA)

Method considered

- All nodes unroll the whole task graph
- They determine tasks they will execute
- They can infer required communications
- No negotiation between nodes (not master-slave)
- Unrolling can be pruned



Node 0 execution

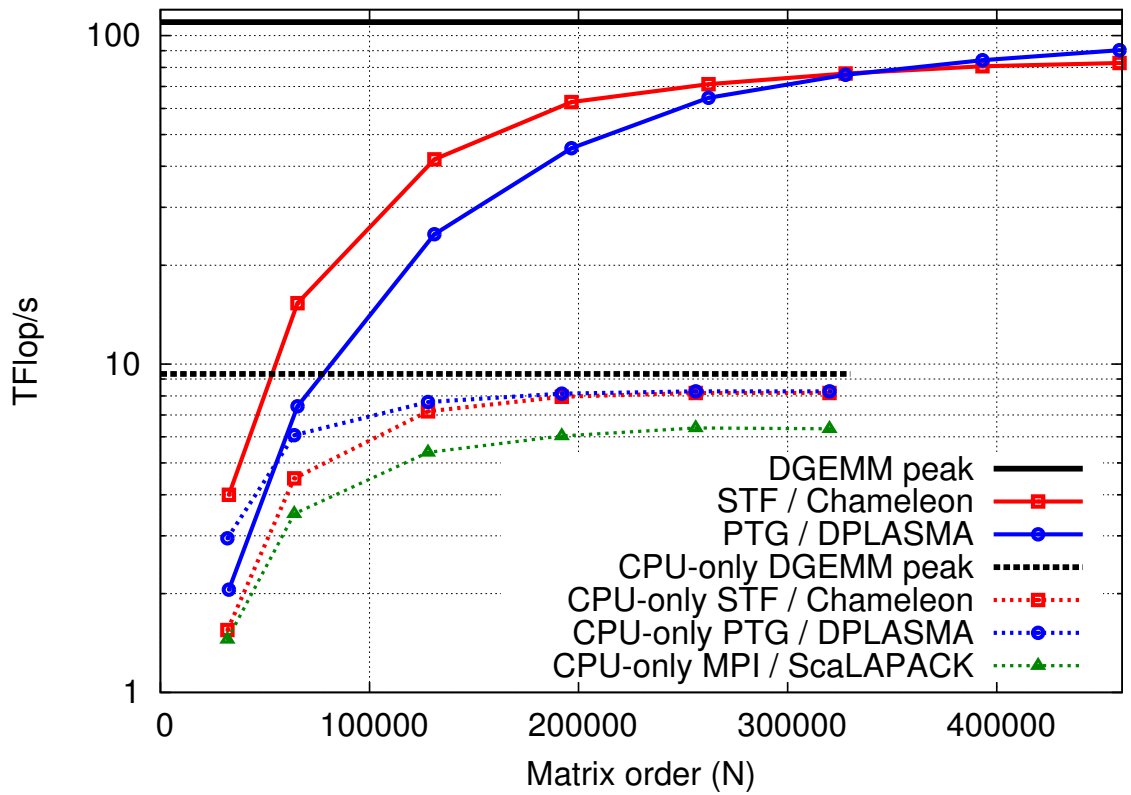


Node 1 execution

Performance

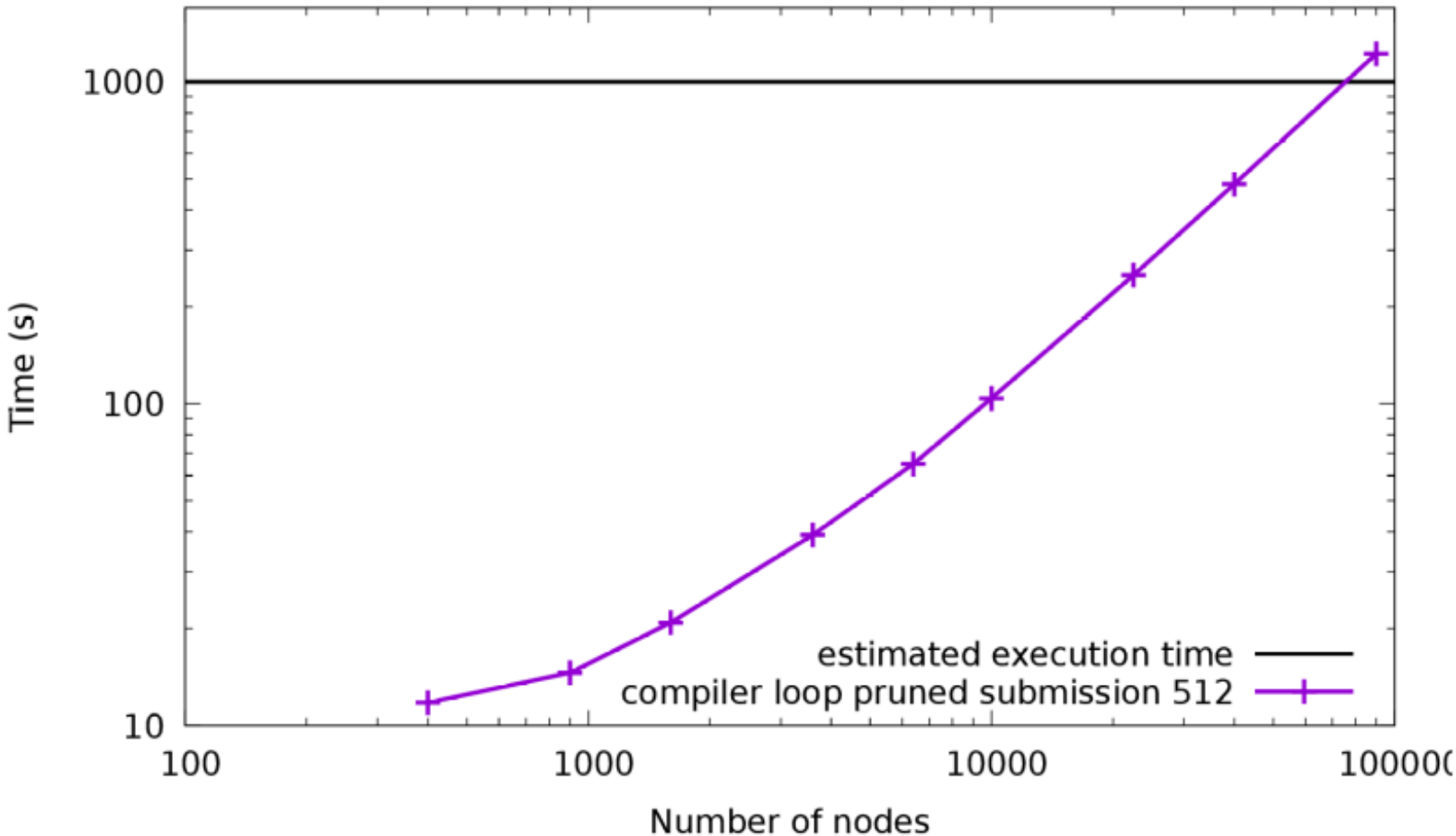
StarPU scheduler used: dmda (Minimum Completion Time type)

- 144 TERA-100 Hybrid nodes
  - collaboration with CEA-CESTA
- CPU: 2 Quad-core Xeon E5620 (per node)
- GPU: 2 NVIDIA Tesla M2090 (per node)



Outline

How far can we go? Submission time vs weak scaling



- Submission time > execution time @ 75000 nodes.
- With these 500 GFlop/s nodes: 37 PFlop/s

Example Manycore Machines

Intel Xeon Phi 7120 (KNL) 64 cores @1.3 GHz, SNC-4, flat MCDRAM

Can we Keep Programming in a Easy STF-like Way?

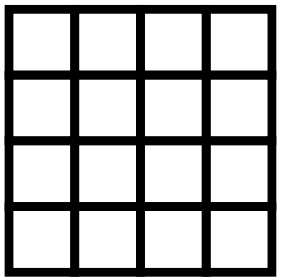
Problem: not only amount of tasks, but also the amount of resources!

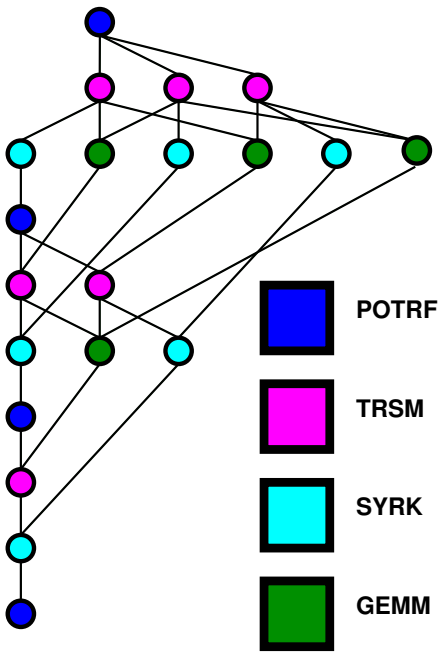
- 1 producer
- 256 consumers

What with the next machines?

How to make schedulers scale?

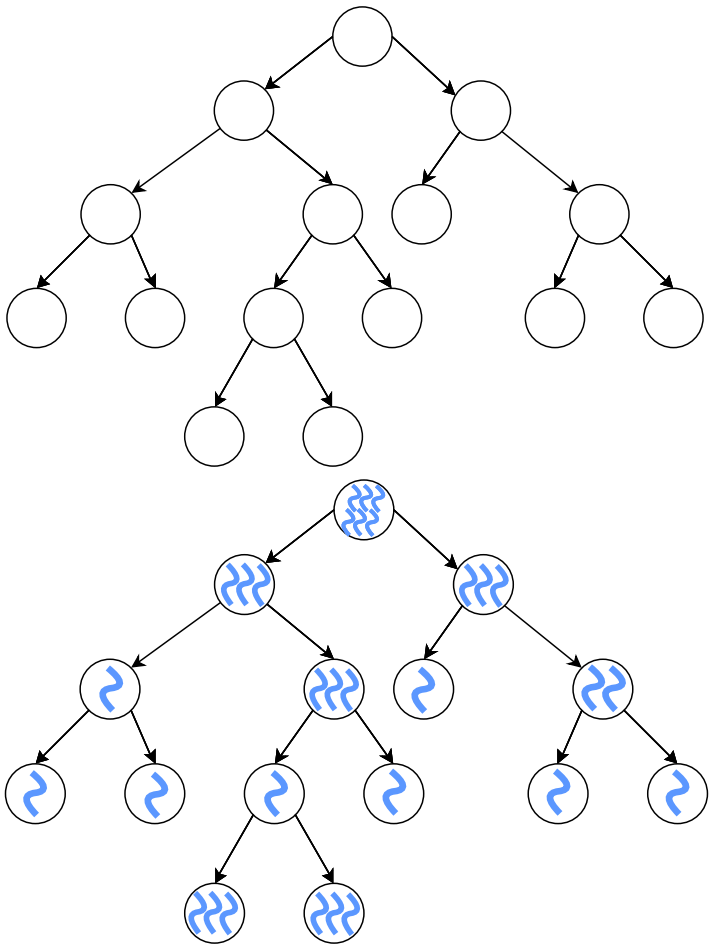
- heft-like schedulers for heterogeneity,
- loop through resources before task submission!





STF With Parallel Tasks : Each Task Can Have Internal Parallelism

Idea: Stop considering resources independently at this scale.



Two parallelism levels

- Task parallelism
- Internal-task parallelism

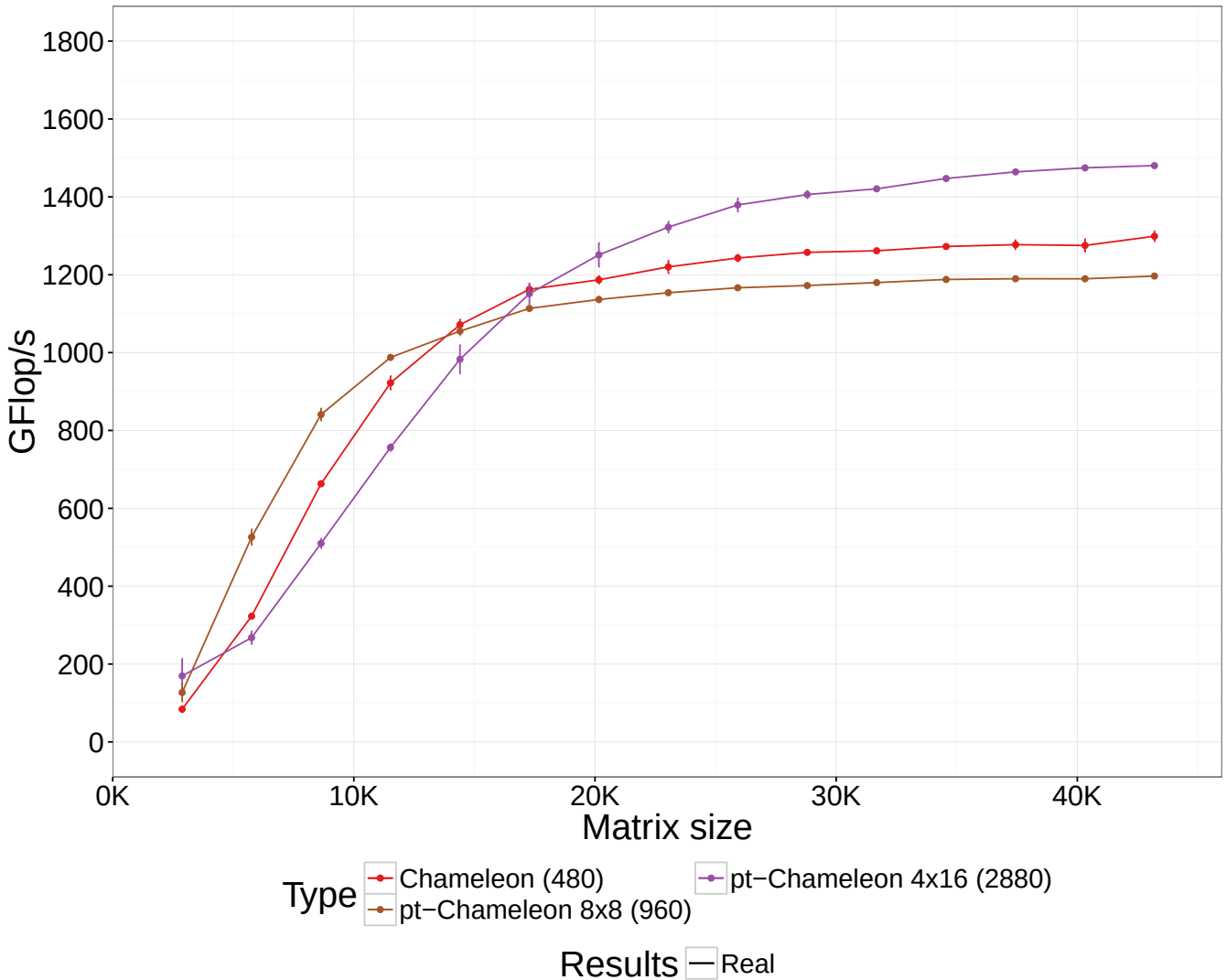
A task has a new parameter: the amount of threads/resources allocated to it.

Here:

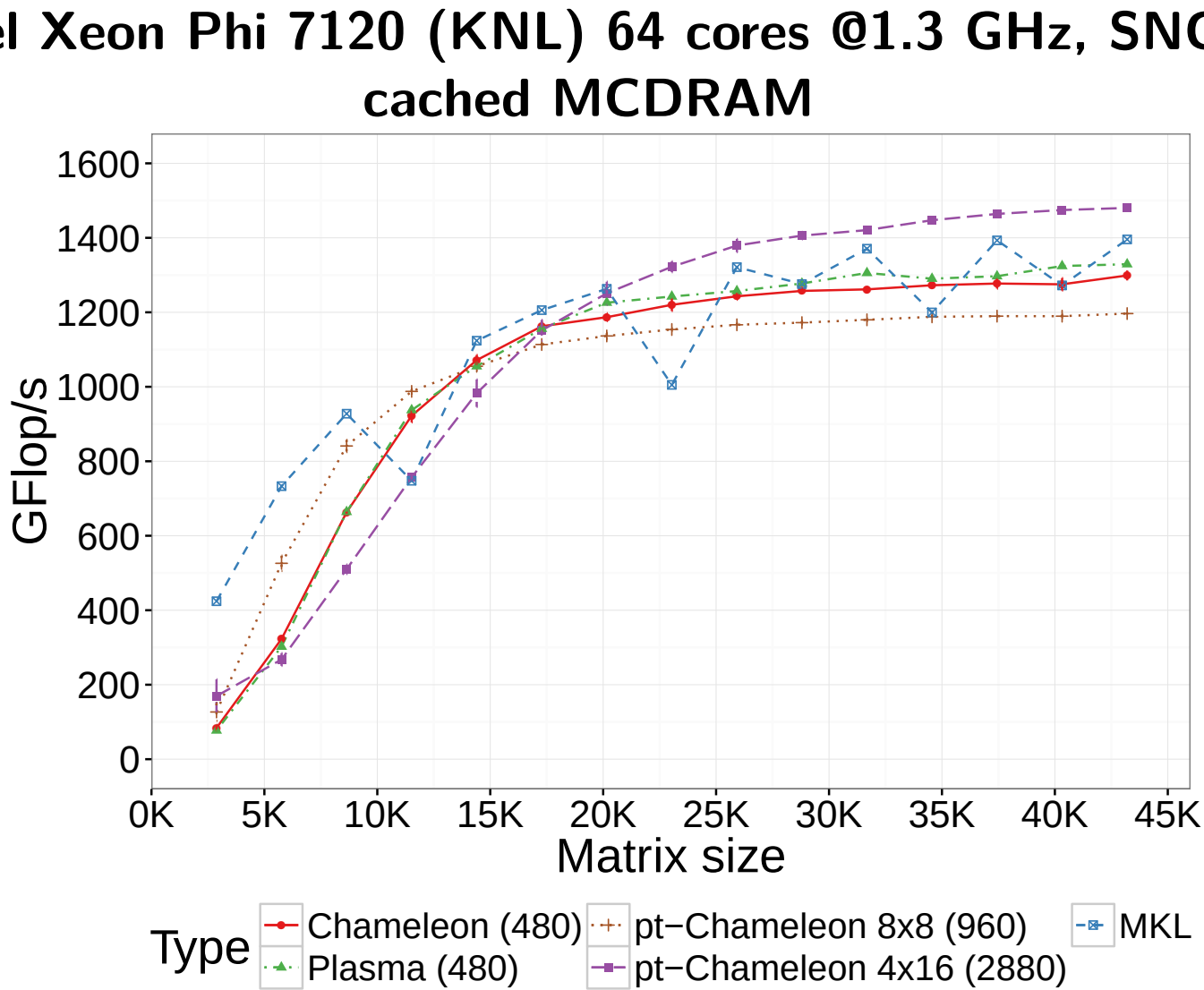
- Task parallelism: StarPU
- Internal parallelism: can be anything. Good candidate: OpenMP (Parallel BLAS and libraries, efficient cache reuse, . . .)

Cholesky Performance on KNL

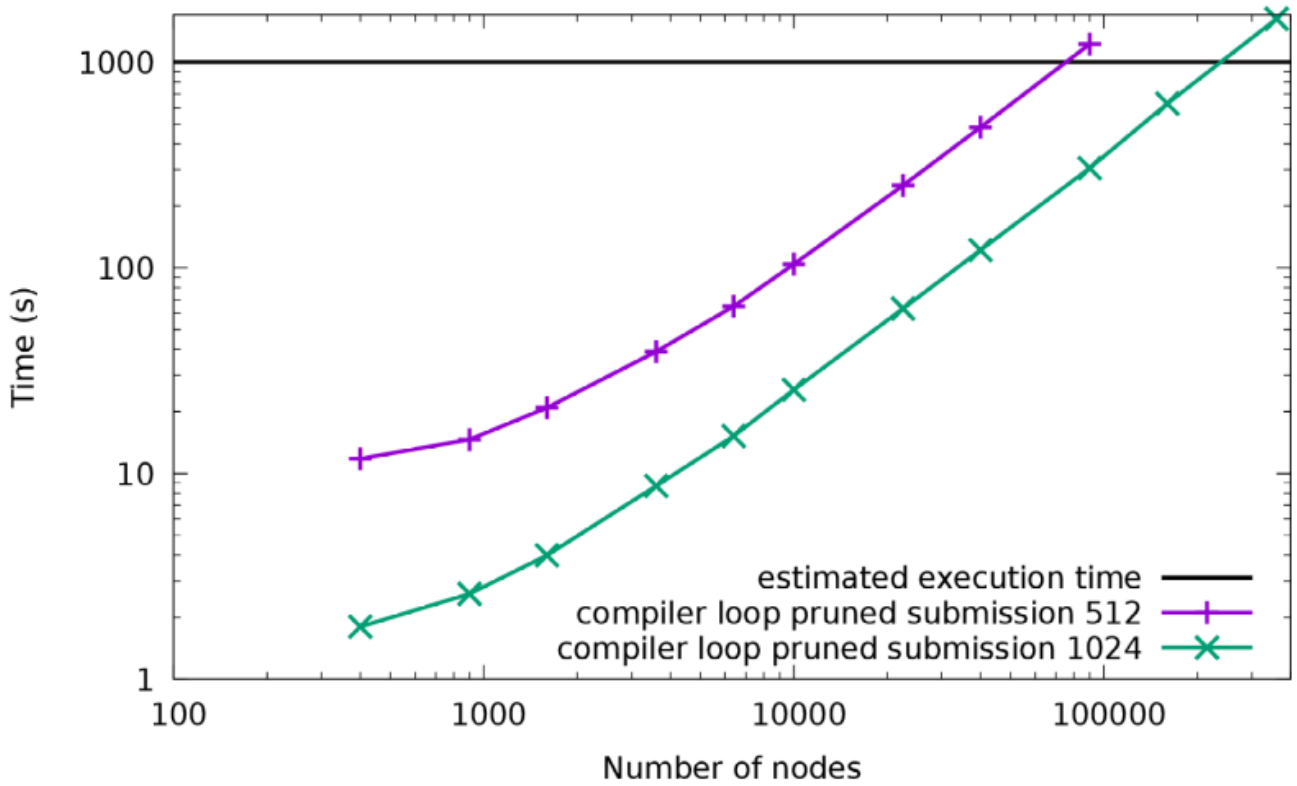
Intel Xeon Phi 7120 (KNL) 64 cores @1.3 GHz, SNC-4, cached MCDRAM



Cholesky Performance on KNL



Going back to the MPI results

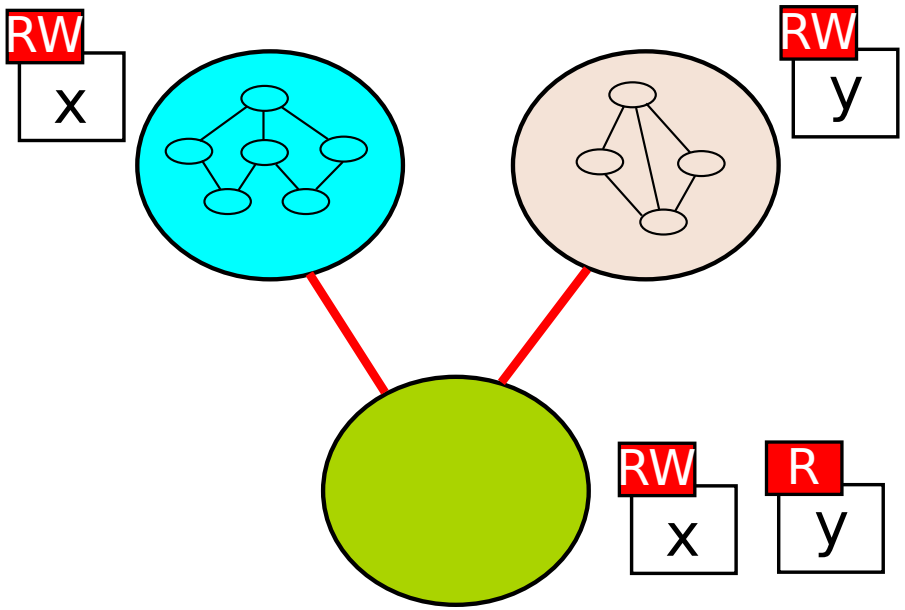


- Increase tile size to 1024 without performance loss
- Less tasks → Less submission time
- Submission time > execution time @ 250000 nodes
- With these old 500 GFlop/s nodes: 125 PFlop/s
- With more recent 4 TFlop/s nodes: 1 EFlop/s

STF with Hierarchical Tasks : StarPU “bubbles”

Idea

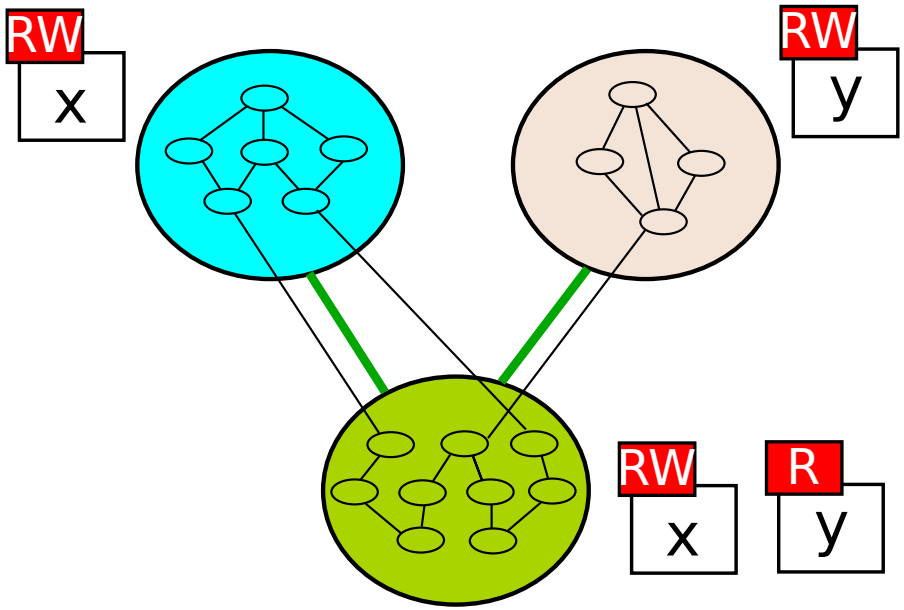
- Tasks which submit tasks
- Metadata as input/output



STF with Hierarchical Tasks : StarPU "bubbles"

Idea

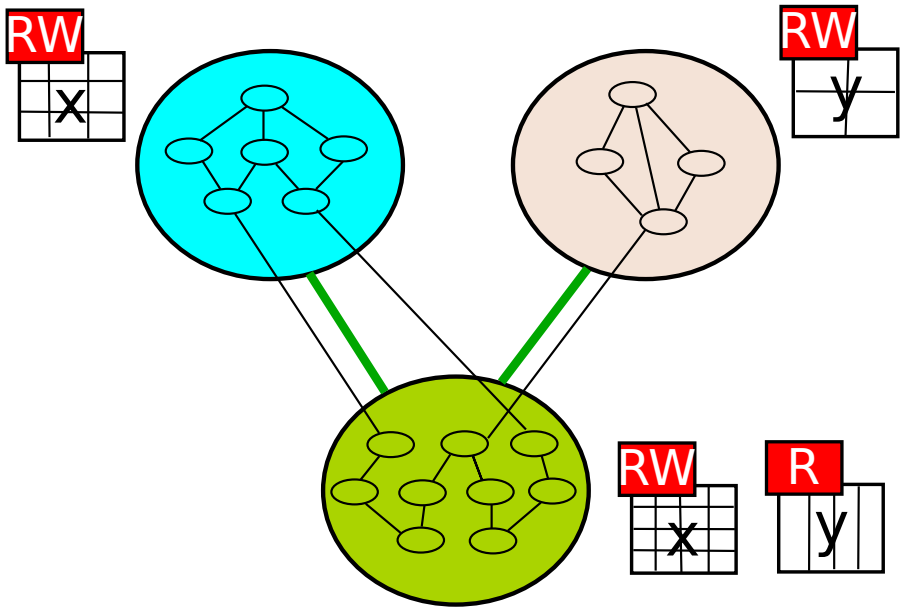
- Tasks which submit tasks
- Metadata as input/output
- Statisfied dependency → submit tasks and discover dependencies



STF with Hierarchical Tasks : StarPU “bubbles”

Idea

- Tasks which submit tasks
- Metadata as input/output
- Statisfied dependency → submit tasks and discover dependencies



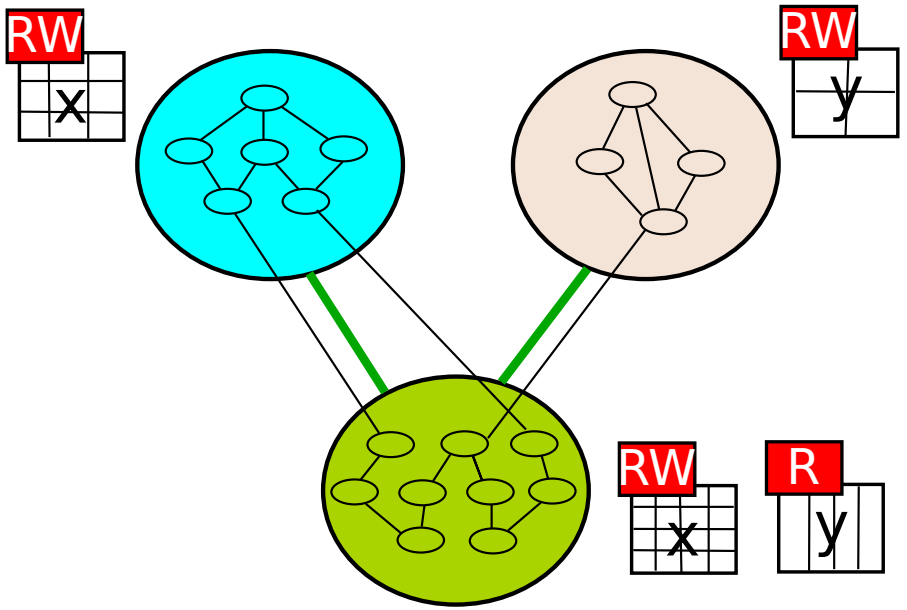
STF with Hierarchical Tasks : StarPU "bubbles"

Idea

- Tasks which submit tasks
- Metadata as input/output
- Statisfied dependency → submit tasks and discover dependencies

Interesting features

- Control granularity
- **Parallel submission**



Currently implemented in `qr_mumps`. Experiments ongoing!

Outline

Conclusion

- StarPU allows easy programming and good performance
- Gives power to the user through the ability to implement schedulers
- Allows (performance) portability of code (e.g. `qr_mumps`)

But there are many challenges ahead on all aspects:

- Retain easy programming and expressivity
- Use of manycore machines
- Efficient heterogeneous schedulers
- Scaling of StarPU-MPI

Showed some proposed solutions with two extensions to our parallelism model

- Parallel tasks
- "SHTF" Sequential Hierarchical Task Flow